**ĐẠI HỌC QUỐC GIA HÀ NỘI**
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**
**KHOA CÔNG NGHỆ THÔNG TIN**

# BÀI TẬP MÔ PHỎNG
# KIẾN TRÚC MÁY TÍNH
# BẰNG LOGISIM

**TÁC GIẢ**
**PGS. TS. Nguyễn Đình Việt (chủ biên)**
**ThS. Lương Việt Nguyên**
**ThS. Nguyễn Hoài Nam**
**ThS. Nguyễn Đức Thiện**

**HÀ NỘI - 2020**

# LỜI NÓI ĐẦU

Máy tính điện tử ra đời vào đầu thập kỷ thứ tư của thế kỷ 20 và đã trở nên hết sức phổ biến trong mọi lĩnh vực hoạt động của con người. Cấu tạo của MTĐT ngày càng tinh vi và phức tạp, bao gồm nhiều thành phần khác nhau. Việc nghiên cứu thiết kế máy tính điện tử là một bài toán khổng lồ và phức tạp, cần không ngừng được cải tiến. Ngày nay người ta thường sử dụng các phần mềm chuyên dụng để thiết kế máy tính, đặc biệt là để thiết kế phần cứng và các thành phần phần mềm ở các mức thấp có chức năng chủ yếu là điều khiển phần cứng.

Tài liệu "Bài tập mô phỏng Kiến trúc máy tính" này được biên soạn nhằm phục vụ cho môn học "Kiến trúc máy tính" ở khoa Công nghệ Thông tin, trường Đại học Công nghệ, Đại học Quốc gia Hà Nội. Các bài thực hành mô phỏng theo tài liệu này bám sát nội dung cuốn sách "Kiến trúc máy tính" của tác giả Nguyễn Đình Việt. Công cụ mô phỏng được sử dụng là Logisim, một bộ mô phỏng mạch điện logic số được sử dụng phổ biến trên thế giới trong lĩnh vực giảng dạy.

Tài liệu gồm 12 bài hướng dẫn thực hành chi tiết để sinh viên có thể tự đọc và làm theo. Mỗi bài có thể được sinh viên hoàn thành trong khoảng thời gian từ 1 đến 2 tiết học. Cuối mỗi bài đều có một vài bài tập thích hợp có gợi ý và đáp án cho sinh viên. Ngoài ra tài liệu còn có 20 bài tập có gợi ý cho sinh viên và đáp án cho giảng viên; Giảng viên có thể sử dụng các bài tập này làm đề tài dự án cho các nhóm sinh viên thực hiện ở nhà.

Sau khi hoàn thành các bài thực hành theo giáo trình này, sinh viên có thể tự thiết kế các mạch logic số từ đơn giản đến tương đối phức tạp, có thể tự xây dựng một máy tính mô phỏng với phần cứng đơn giản và một vi chương trình với chức năng của bộ thông dịch, làm cho chiếc máy tính với phần cứng đơn giản này có khả năng thực hiện các chương trình ở mức "Máy thông thường" tương tự như tất cả các máy tính hiện đại khác.

Tài liệu này tuy đã được biên soạn và sử dụng liên tục trong nhiều năm học, bắt đầu từ năm học 2010-2011, nhưng chắc không tránh khỏi việc vẫn còn thiếu sót. Nhóm tác giả chân thành mong muốn nhận được các ý kiến đóng góp về mọi mặt của các bạn đồng nghiệp, các bạn sinh viên và bạn đọc.

Các ý kiến đóng góp xin gửi về địa chỉ:

Nguyễn Đình Việt

Khoa Công nghệ Thông tin, Trường Đại học Công nghệ, ĐHQGHN

144 Đường Xuân Thuỷ, Quận Cầu giấy, Hà nội

Email: vietnd@vnu.edu.vn

**MỤC LỤC**

## **ĐÁP ÁN CÁC BÀI TẬP (DÀNH CHO GIẢNG VIÊN)**Error! Bookmark not defined.

# DANH SÁCH CÁC HÌNH

# Bài số 1: Học sử dụng Logisim theo trợ giúp (Help)

## 1.1. Các yêu cầu và nội dung chính

- Biết cách cài đặt và cho chạy chương trình mô phỏng
- Biết cách sử dụng trợ giúp (Help) của Logisim
- Biết các bước tạo một mạch logic số đơn giản
- Biết cách lưu giữ (save) các mạch điện đã tạo ra vào file và mở (open).

## 1.2. Giới thiệu bộ mô phỏng Logisim

### 1.2.1 Logisim và tác giả

Logisim là một công cụ đào tạo dùng để thiết kế và mô phỏng các mạch điện logic số. Với một giao diện thanh công cụ đơn giản và khả năng mô phỏng các mạch điện, Logisim đủ đơn giản để làm cho việc học các khái niệm cơ bản nhất liên quan đến các mạch điện logic được thuận tiện. Với khả năng xây dựng được các mạch điện lớn hơn từ các mạch con (subcircuit) nhỏ hơn và khả năng vẽ các búi dây bằng chỉ một động tác kéo chuột, Logisim có thể được sử dụng để thiết kế và mô phỏng toàn bộ một CPU phục vụ cho mục đích đào tạo.

Sinh viên rất nhiều trường đại học trên toàn thế giới đã và đang sử dụng Logisim cho nhiều mục đích khác nhau, bao gồm:

- Làm một mô-đun trong phần giới thiệu tổng quan về khoa học máy tính
- Làm một đơn vị kiến thức trong các khóa về tổ chức máy tính ở mức năm thứ hai bậc đại học
- Cho sinh viên học tập trong cả một học kỳ trong các khóa học kiến trúc máy tính ở mức nâng cao.

Tác giả của Logisim là  giáo sư Carl Burch; Có thể tìm thấy các thông tin mà tác giả tự giới thiệu tại địa chỉ URL: http://www.cburch.com/personal/about.html. Phần mềm Logisim được tác giả đăng ký bản quyền năm 2005. Đây là một phần mềm tự do; người dùng có thể phân phát và/hoặc sửa đổi nó miễn là tuân theo các điều kiện của Giấy phép công cộng GNU (GNU General Public License) được Tổ chức  phần mềm tự do (Free Software Foundation) công bố.

### 1.2.2 Cài đặt bộ mô phỏng Logisim

Có thể download các bản Logisim cho Windows, Linux, MacOS tại các địa chỉ sau:

- Bản cho Windows mới nhất tính đến 8/2019 là version 2.7.1 được phát hành ngày 21/3/2011, tên file là logisim-2-7-1-es-en-br-de-win.exe, có thể download tại địa chỉ: https://logisim.en.uptodown.com/windows/download.
- Bản cho Linux (và nhiều OS khác) mới nhất tính đến 8/2019 là version 2.7.1 được phát hành ngày 29/4/2013, tên file là logisim-generic-2.7.1.jar, có thể download tại địa chỉ: https://sourceforge.net/projects/circuit/

**Chạy Logisim trong môi trường Windows:**

Nếu máy tính đã cài đặt java, thì có thể chạy Logisim ngay như chạy các file chương trình khả thi bình thường khác. Nếu chưa, thì phải cài java rồi mới chạy được Logisim.

**Chạy Logisim trong môi trường Linux (Ubuntu):**

Ubuntu từ phiên bản 12.04 trở đi coi Logisim là một trong các gói phần mềm chuẩn, có thể tìm và cài đặt qua các ứng dụng như Synaptic Package Manger, Ubuntu Software Center v.v. Các ứng dụng này sẽ tự động tìm và cài đặt các gói phần mềm thư viện cần thiết cho Logisim.

Với các phiên bản Ubuntu trước đó, cần phải cài đặt java trước mới có thể chạy được Logisim. Chúng tôi đã cài sun-java6-jre trong Ubuntu 10.04 và Ubuntu Remix và không gặp trở ngại nào. Có thể dùng Synaptic Package Manger để tìm và cài bản java này. Chạy Logisim như sau: java –jar logisim-2.3.5.jar [Enter]

## 1.3. Các bước tạo một mạch logic số đơn giản

Thực hiện các bước theo "Help" của Logisim (Help \ Tutorial \ Beginner's Tutorial). Trong "Bài tập mô phỏng số 1", sinh viên cần đọc và làm theo 5 bước (step 0 – step 4) được trình bày trong mục "Beginner's Tutorial". Các nội dung của mục 1.3 dưới đây được lược dịch từ mục nói trên.

### Chào mừng đến với Logisim

Logisim cho phép chúng ta thiết kế và mô phỏng các mạch điện số. Nó được tạo ra để làm một công cụ học tập, giúp người học tìm hiểu về sự làm việc của các mạch điện.

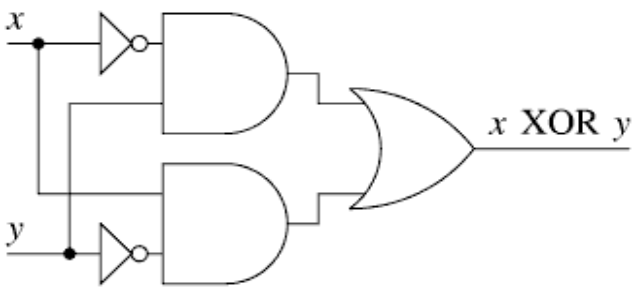
Để thực hành sử dụng Logisim, chúng ta bắt đầu bằng việc xây dựng một mạch XOR, đó là một mạch điện có 2 đầu vào (chúng ta sẽ gọi là x và y) và một đầu ra, đầu ra có giá trị 0 nếu hai đầu vào có giá trị giống nhau và có giá trị 1 nếu chúng có giá trị khác nhau.

Bảng chân lý (Truth table) dưới đây minh họa chức năng của mạch XOR.

Bảng 1.1

| x | y | x XOR y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Chúng ta có thể thiết kế mạch điện như vậy trên giấy, như trên hình 1.1.

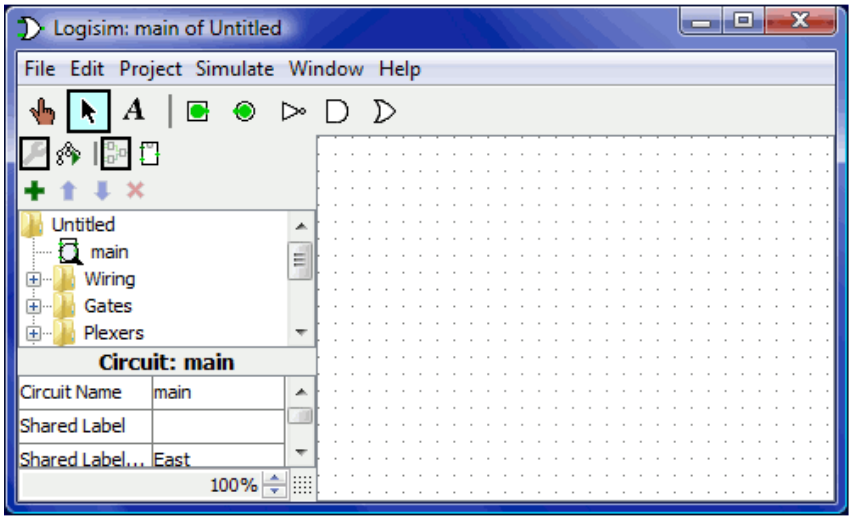



Hình 1.1 Mạch điện thực hiện chức năng XOR

Tuy nhiên, vì đây là mạch điện trên giấy nên ta có thể không chắc có đúng hay không. Để kiểm tra chúng ta sẽ dùng Logisim để vẽ mạch điện này và thử. Ngoài việc thử được, mạch điện vẽ bằng Logisim có thể trông đẹp hơn mạch điện vẽ trên giấy.

### 1.3.1 Bước 0: Tự định hướng (Orienting yourself)

Khi mới bắt đầu chạy Logisim, chúng ta sẽ thấy một cửa sổ như ở hình 1.2 dưới đây. Một số chi tiết trên cửa sổ mà các bạn nhìn thấy trên máy tính của mình có thể hơi khác vì hệ thống máy tính của các bạn cũng như phiên bản (version) Logisim khác với hệ thống và phiên bản được chúng tôi sử dụng.




Hình 1.2 Cửa sổ của Logisim khi mới khởi động

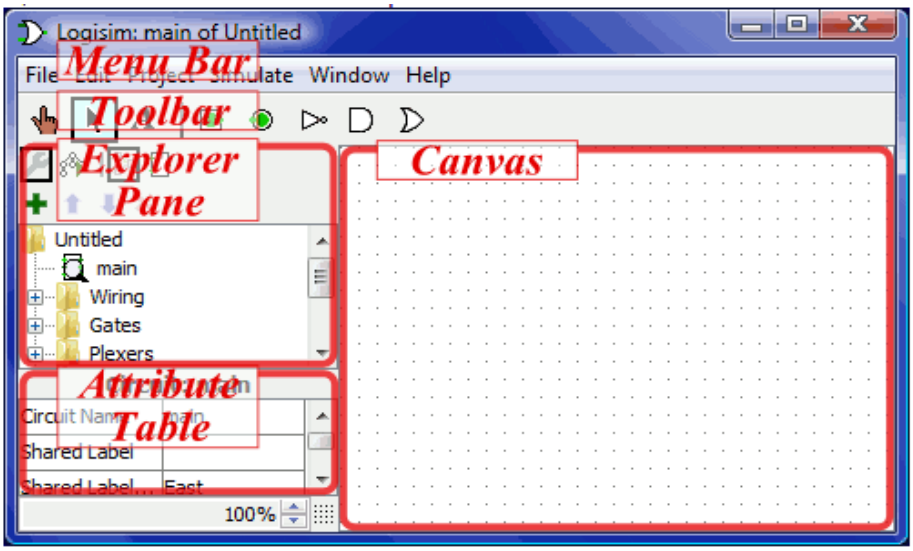
Cửa sổ của Logisim bao gồm 5 phần, được mô tả trên hình 1.3:

**Menu Bar:** Phần trên cùng là Menu Bar (Thanh thực đơn), gồm các mục chọn: File, Edit, Project, Simulate, Window và Help. Tên gọi và ý nghĩa của Menu Bar cũng như các mục chọn trong đó tương tự như trong rất nhiều phần mềm thông dụng khác, chắc không cần giải thích.

**Toolbar:** Bên dưới Menu Bar là Toolbar (Thanh công cụ), chứa các công cụ được thể hiện bằng các biểu tượng, cụ thể như sau:

- Poke tool (biểu tượng là bàn tay với ngón trỏ hướng lên trên)
- Edit Tool (biểu tượng là mũi tên hướng lên trên và chếch sang trái)
- Text Tool (biểu tượng là ký tự A hoa nghiêng và đậm)
- Add input pin (biểu tượng là hình vuông, bên trong có chấm xanh có một đầu chìa ra ngoài hình vuông)
- Add output pin (biểu tượng là hình tròn, bên trong có chấm xanh, có một đầu chìa ra ngoài hình tròn)
- Add NOT gate (biểu tượng là cổng NOT)
- Add AND gate (biểu tượng là cổng AND)
- Add OR gate (biểu tượng là cổng OR)

Người sử dụng dùng các công cụ này để tạo ra các mạch điện cũng như chỉnh sửa và điều khiển việc mô phỏng. Tất cả các công cụ trong Toolbar cũng có thể lấy để sử dụng từ  Explorer Pane sẽ được trình bày ngay dưới đây.




Hình 1.3 Năm phần chính trên cửa sổ của Logisim

**Explorer Pane (Ô cửa thăm dò):** Tại đây chúng ta có thể lấy tất cả các phần tử mạch điện mà Logisim đã xây dựng sẵn để đưa vào mạch điện của mình ở Canvas (Bức vẽ). Ngoài ra chúng ta cũng có thể lấy tất cả các công cụ có trong Toolbar và một số công cụ khác nữa, đó là Selection Tool, Wiring Tool, Menu Tool và Label.

**Attribute Table (Bảng thuộc tính):** Bên dưới Explorer Pane là Attribute Table. Đây là nơi người sử dụng có thể lựa chọn theo yêu cầu của mình một số giá trị hoặc thuộc tính, hoặc cách thể hiện bằng đồ họa của hầu hết các phần tử mạch điện có sẵn trong thư viện của Logisim. Chúng ta có thể chọn phần tử cần thay đổi thuộc tính ở Explorer Pane hoặc ở trên Canvas.

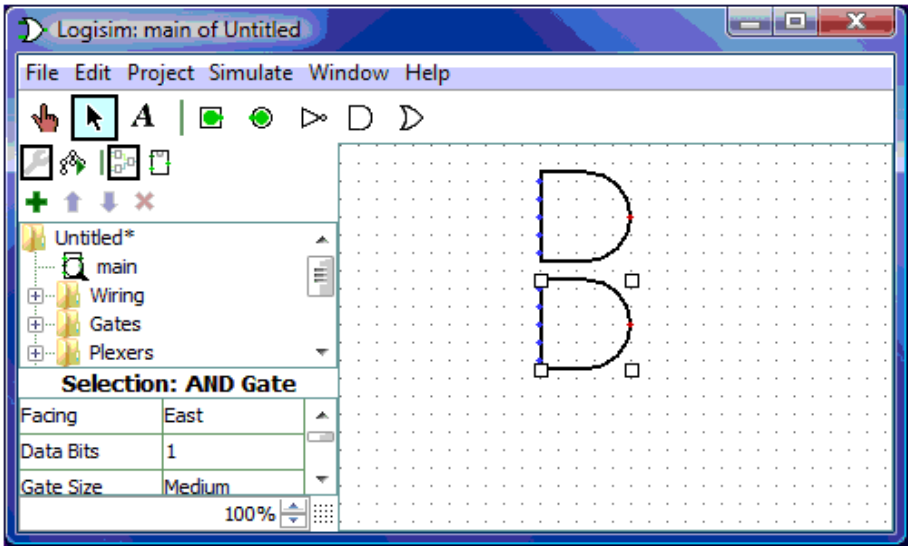
**Canvas (Bức vẽ):** Đây có thể coi là cửa sổ làm việc của người sử dụng. Tại đây người sử dụng sẽ lắp ráp các phần tử mạch điện thành mạch điện theo yêu cầu của mình, sau đó sẽ thực hiện mô phỏng (simulate) để kiểm tra tính đúng đắn của mạch điện đó.

Chúng ta sẽ dần dần sử dụng thành thạo các phần tử mạch điện, các công cụ và các chức năng căn bản khác của Logisim trong các bài học và trong quá trình làm các bài tập trong tài liệu này.

### 1.3.2 Bước 1: Bổ sung các cổng (Adding gates)

Chúng ta sẽ thử xây dựng mạch điện như trên hình 1.1. Theo lời khuyên của tác giả Logisim là nên đưa các cổng vào bức vẽ (Canvas) trước làm bộ khung cho mạch điện, sau đó mới nối chúng với nhau. Việc đầu tiên chúng ta làm là đưa vào 2 cổng AND bằng cách kích vào biểu tượng của cổng AND (biểu tượng của nó nằm trước biểu tượng cuối cùng trong Toolbar. Khi ta di chuyển con trỏ Edit Tool đến chỗ có biểu tượng này thì Logisim hiện dòng chữ "Add AND gate (Ctrl-7") sau đó kích vào chỗ trong Canvas mà chúng ta muốn đặt cổng AND này. Sau đó ta lại kích vào biểu tượng "Add AND gate" rồi kích vào chỗ bên dưới cổng AND đã có ở đó trong Canvas

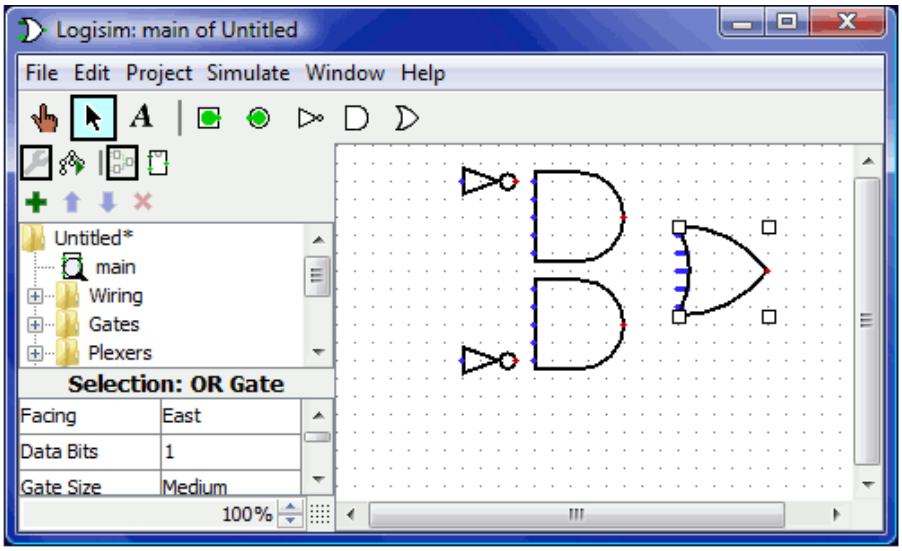
để đặt cổng AND thứ hai. Hình 1.4 dưới đây là kết quả mà chúng ta nhận được trong cửa sổ của Logisim.




Hình 1.4 Đặt 2 cổng AND vào trước làm bộ khung cho mạch điện

Chú ý: 5 chấm xanh ở mặt trái của cổng AND là các đầu vào (input) của nó, số lượng đầu vào có thể thay đổi được trong miền từ 1 đến 32 bằng cách sử dụng "Attribute table", mục Number Of Inputs. Các chân không cần sử dụng không làm thay đổi chức năng của cổng AND, nghĩa là chúng ta coi như không có, không cần nối với phần tử mạch điện nào.

Tiếp theo chúng ta sẽ bổ sung thêm các cổng còn lại khác. Đầu tiên cần kích vào "OR tool" trên Toolbar, rồi kích vào vị trí trên cửa sổ Canvas mà chúng ta muốn đặt cổng OR. Bằng cách tương tự như vậy, chúng ta bổ sung thêm 2 cổng NOT vào cửa sổ Canvas. Trên cửa sổ Canvas chúng ta có thể thấy mạch điện đang xây dựng như trên hình 1.5.



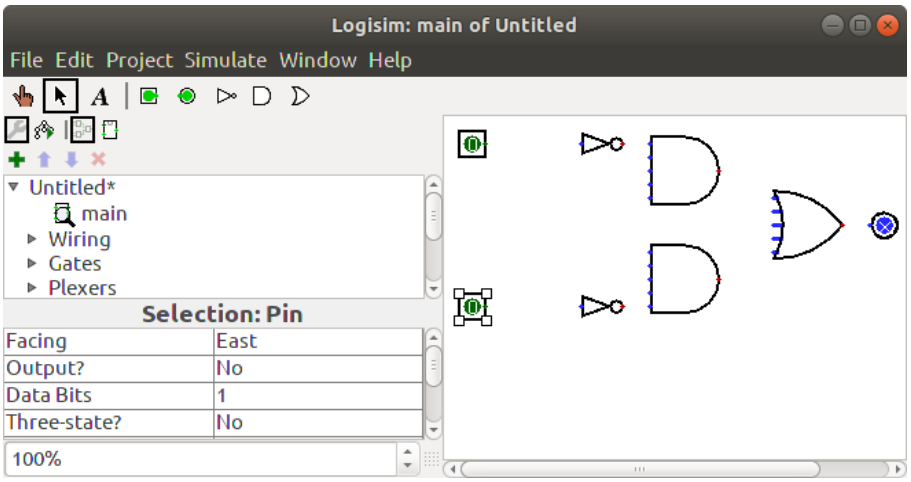

Hình 1.5 Sau khi đặt 2 cổng AND vào trước, bổ sung 2 cổng NOT và 1 cổng OR

Nếu muốn xê dịch vị trí một phần tử mạch điện nào đó, chúng ta cần dùng công cụ "Edit tool" (biểu tượng là mũi tên), hãy kích chuột vào đó rồi trỏ vào phần tử cần dịch chuyển, rồi kéo và

thả (drag and drop) ở vị trí mong muốn. Nếu chúng ta muốn xóa bỏ một phần tử nào đó thì chỉ cần kích vào phần tử đó rồi chọn lệnh Delete ở Edit menu hoặc ấn phím Delete trên bàn phím, hoặc nhấn nút chuột phải lên phần tử đó rồi chọn Delete ở bảng chọn sẽ hiện ra.

Mặc dù có thể để đầu ra của mỗi cổng NOT dính (nối) vào một đầu vào của cổng AND, tuy nhiên nhìn trên hình mạch điện với các phần tử dính sát với nhau có thể không đẹp hoặc không rõ ràng. Vì vậy, người ta thường để các phần tử của mạch điện cách nhau một khoảng cách nhất định và dùng phần tử Wire (dây nối) để nối các điểm đầu vào hay đầu ra (input/output) của các phần tử mạch điện với nhau.

Tiếp theo chúng ta bổ sung 2 phần tử input và 1 phần tử output vào sơ đồ bằng cách lần lượt sử dụng "Input tool" và "Output tool" trên Toolbar để đặt chúng cạnh 2 đầu vào của 2 cổng NOT và đầu ra của cổng OR. Hình 1.6 là mạch điện mà chúng ta đang xây dựng đến lúc này.




Hình 1.6 Mạch điện đang xây dựng sau khi đã bổ sung 2 phần tử input và 1 phần tử output

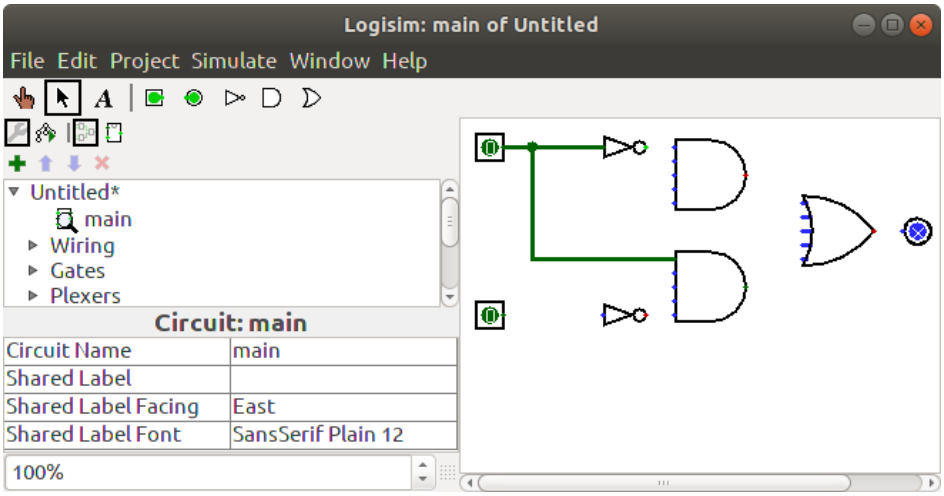
Nếu chú ý chúng ta có thể nhận thấy trên màn hình Logisim, mỗi khi chúng ta đặt xong một phần tử mạch điện trên cửa sổ Canvas thì Logisim lại chuyển về Edit tool để cho chúng ta có thể di chuyển phần tử vừa đặt hoặc nối dây phần tử đó với một phần tử đã có nào đó. Nếu chúng ta muốn đặt vào mạch điện một phần tử vừa đặt vào xong, thì có thể dùng chức năng copy bằng cách nhấn tổ hợp phím Control-D (Duplicate) (Một số loại máy tính có thể dùng tổ hợp phím khác, chẳng hạn với dòng máy Mac cần dùng tổ hợp phím Command-D).

### 1.3.3 Bước 2: Bổ sung thêm dây nối (Adding wires)

Sau khi thực hiện xong bước 1 chúng ta đã có tất cả các phần tử mạch điện trên Canvas, giờ là lúc chúng ta chúng ta nối chúng lại với nhau bằng dây (wire), đây là một loại phần tử có sẵn trong thư viện của Logisim. Chúng ta dùng Edit Tool để nối dây, khi con trỏ được rê đến điểm có thể nối dây (input, output hoặc một đầu dây trên Canvas) Logisim sẽ hiện một vòng tròn nhỏ màu xanh lá cây xung quanh điểm đó, ta nhấn nút chuột để đánh dấu một đầu dây rồi rê chuột để kéo dây đến điểm muốn kết thúc rồi nhả nút chuột.

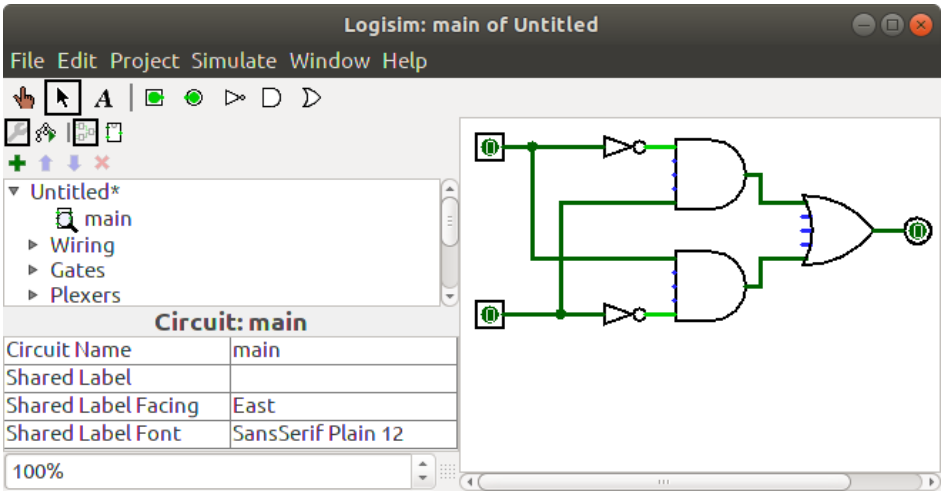
Logisim khá thông minh trong việc nối dây: nếu điểm cuối của một dây nằm trên một dây khác thì hai dây này được tự động nối với nhau; nếu một đầu dây nằm trùng một đầu ra hay đầu vào của một cổng thì đầu dây đó được tự động nối với đầu ra hay đầu vào đó. Chúng ta cũng có thể kéo dài hoặc rút ngắn một dây bằng cách dùng Edit Tool kéo một đầu của nó.

Dây trên cửa sổ Canvas của Logisim chỉ có thể chạy theo chiều ngang hoặc chiều đứng. Để nối phần tử Input bên trên với đầu vào của cổng NOT, nối với đầu vào bên trên của cổng AND bên dưới và nối đầu ra của cổng not với một đầu vào của cổng AND bên trên, chúng ta cần chạy 2 đoạn dây, như minh họa trên hình 1.7.




Hình 1.7 Bổ sung 2 đoạn dây để nối phần tử Input bên trên với các điểm cần thiết

Khi chúng ta vẽ các đường dây, chúng ta có thể thấy có đường dây màu xanh (blue), có đường dây màu xám (gray). Màu xanh trên dây (hoặc trên các điểm đầu vào, đầu ra) cho biết rằng, giá trị logic truyền trên đó là không xác định (unknown), còn màu xám cho biết rằng dây không được nối với đâu cả. Trong khi còn đang xây dựng mạch điện thì vấn đề một dây có màu xanh hay xám không quan trọng. Tuy nhiên khi chúng ta hoàn thành mạch điện, không một dây nào được phép có màu xanh hoặc xám, trừ đầu vào không được sử dụng của cổng có thể xanh (blue) cũng không sao.




Hình 1.8 Các dây có màu xanh lá cây nhạt hoặc xanh xẫm - không có gì sai

Nếu chúng ta thấy trên mạch điện mà mình cho là đã xây dựng xong, nhưng vẫn có dây có màu xanh (blue) hoặc xám (gray), thì điều có có nghĩa là có gì đó sai. Điều quan trọng là chúng ta phải nối dây đến đúng chỗ cần thiết. Logisim vẽ một chấm nhỏ trên các phần tử mạch điện để

chỉ ra rằng, điểm đó cần nối với dây. Khi thực hành tiếp, chúng ta sẽ thấy các chấm chuyển màu từ xanh (blue) sang xanh nhạt (light blue) hoặc xanh xẫm.

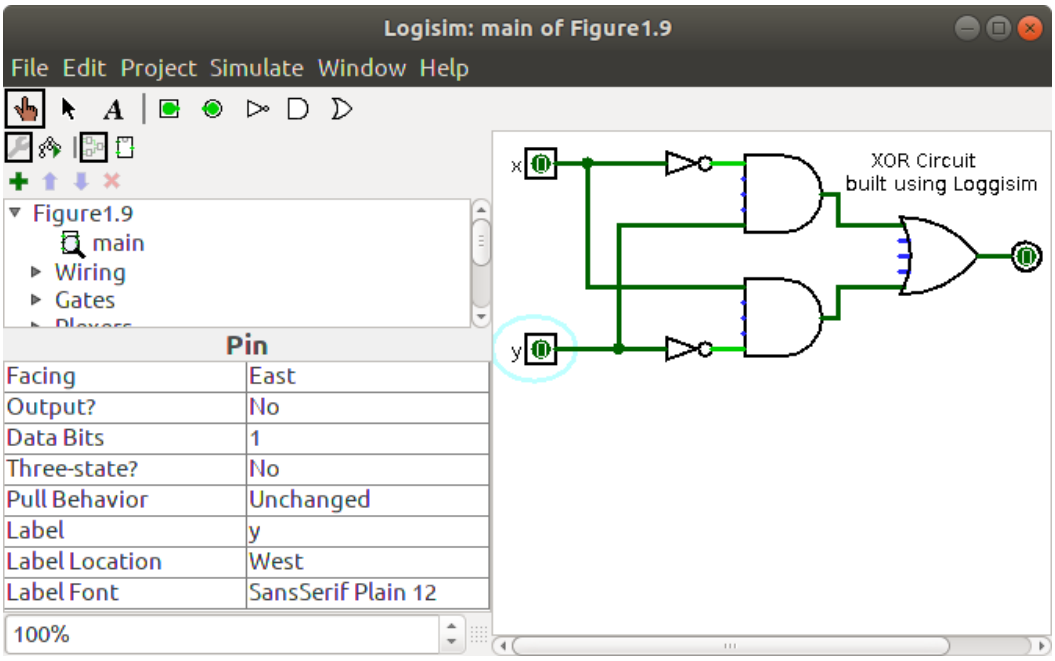
### 1.3.4 Bước 3: Bổ sung chữ (Adding text)

Chữ không liên quan gì đến việc thực hiện các chức năng của mạch điện mà chúng ta xây dựng. Chữ được bổ sung thêm vào chỉ có tác dụng giống như "comment" trong các chương trình máy tính, giúp người xem mạch điện dễ hiểu, kể cả cho chính người thiết kế ra nó. Để đưa text vào mạch điện trên Canvas, chúng ta có thể làm theo hai cách sau:

Cách thứ nhất: Kích nút chuột vào phần tử mạch điện mà chúng ta muốn dùng text làm nhãn, thí dụ phần tử Pin, Logisim sẽ hiện bảng "Selection: Pin" để chúng ta gõ text vào dòng có nhãn "Label", ngoài ra có thể lựa chọn một số thuộc tính của đoạn text (Facing, Label Location, Label Font). Nên dùng cách thứ nhất này khi cần ghi text làm nhãn cho các cổng cũng như các phần tử mạch điện có sẵn trong thư viện của Logisim. Theo cách này, khi chúng ta di chuyển phần tử mạch điện, text sẽ di chuyển theo.

Cách thứ hai: Kích nút chuột vào "Text tool" (biểu tượng là chữ "A" trên Toolbar) rồi kích vào chỗ trên Canvas nơi chúng ta muốn đưa đoạn text vào từ bàn phím. Nên dùng cách thứ hai này khi cần ghi các đoạn text tương đối dài để mô tả mạch điện hoặc hướng dẫn sử dụng v.v.

Trên hình 1.9 là màn hình Logisim khi chúng ta áp dụng cách thứ nhất để thực hiện gán nhãn (label) "y" cho phần tử input bên dưới của mạch điện trên Canvas.




Hình 1.9 Màn hình Logisim khi muốn gán nhãn "y" nằm bên trái một phần tử input

### 1.3.5 Bước 4: Kiểm thử mạch điện (Testing your circuit)

Bước cuối cùng là kiểm thử mạch điện mà chúng ta vừa xây dựng để đảm bảo rằng nó thực sự thực hiện đúng chức năng mà chúng ta muốn.

Hãy nhìn mạch điện trên cửa sổ Canvas của Logisim (như hình 1.9): cả 2 lối vào (2 phần tử input) x và y đều chứa mức 0, đầu ra (phần tử output) cũng có giá trị 0. Điều này cho chúng ta thấy rằng mạch điện đã tính ra kết quả 0 khi cả 2 lối vào cùng bằng 0.

Bây giờ chúng ta có thể thử thiết lập các tổ hợp giá trị lối vào khác, bằng cách sử dụng công cụ Poke tool (poke: chọc bằng ngón tay). Mỗi lần kích vào phần tử input, giá trị của nó sẽ đảo (từ 0 thành 1 hoặc từ 1 thành 0). Với 4 tổ hợp giá trị (2 bit) lối vào khác nhau, chúng ta có thể quan sát thấy giá trị đầu ra ở phần tử output như ở bảng 1.2. Nếu kết quả nhận được đúng như vậy, có nghĩa là mạch điện đầu tiên mà chúng ta thiết kế đã làm việc và làm việc đúng.

Bảng 1.2 Bảng chân lý của hàm XOR

| x | y | x XOR y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Để lưu kết quả công việc, chúng ta vào thực đơn File, chọn lệnh Save hoặc Save As. Tất nhiên chúng ta cũng có thể lưu trữ kết quả ra giấy bằng lệnh Print hoặc Export Image ở thực đơn File.

Bây giờ là lúc chúng ta có thể ra khỏi (exit) Logisim vì đã học xong tài liệu hướng dẫn (tutorial) này. Tất nhiên, để xây dựng được các mạch điện phức tạp hơn, chúng ta chắc sẽ phải xem toàn bộ trợ giúp (Help).

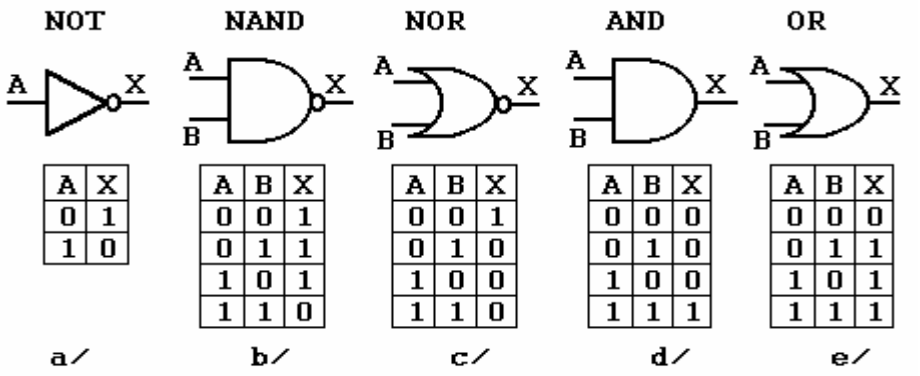
Logisim là một chương trình rất mạnh, cho phép chúng ta có thể xây dựng và kiểm thử các mạch điện rất lớn. Việc xây dựng mạch điện nói trên chỉ là một thí dụ rất đơn giản trong quá trình từng bước học sử dụng Logisim.

## 1.4. Bài tập (3 bài)

**Các bài tập này theo sát nội dung giáo trình Kiến trúc máy tính; Khi làm cần chú ý:**

- Đầu ra của 1 cổng có thể nối với đầu vào của 1 cổng khác. Đối với các đầu vào/ra không nối với các đầu vào/ra của các cổng khác, mỗi đầu vào cần nối với một phần tử "input" và mỗi đầu ra cần nối với phần tử "output".
- Để thử lại chức năng, sử dụng công cụ "Poke tool" trỏ vào các phần tử "input" và kích nút chuột bên trái để đảo giá trị của input, đồng thời quan sát giá trị của phần tử "output".
- Sau khi đã tạo và thử các mạch logic, cần "save" vào bộ nhớ ngoài (ổ đĩa cứng, ổ SSD, ổ đĩa USB...), sau đó thử "open" để xem lại. Tên file lưu mạch điện theo các hình vẽ trong tài liệu này nên đặt theo tên hình vẽ cho dễ nhớ; Thí dụ, với mạch điện trên hình 1.10, tên file nên đặt là "hinh1-10.circ" hoặc "figure1-10.circ" v.v.

**Bài tập 1.1:** Mô phỏng sự hoạt  động của các cổng logic đơn giản nhất trên hình 1.10 (Hình 3-02, GT KTMT).




| A | X |
|---|---|
| 0 | 1 |
| 1 | 0 |

| A | B | X |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

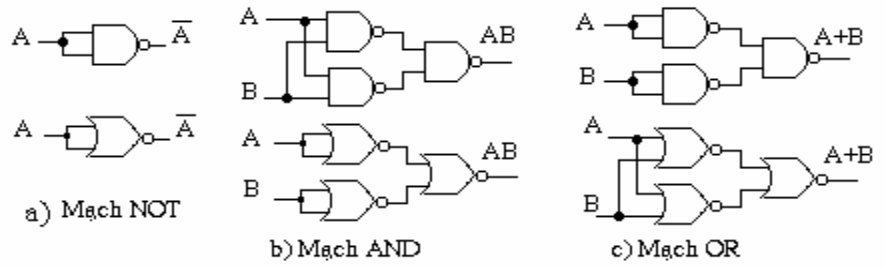
| A | B | X |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

| A | B | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| A | B | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Hình 1.10 Ký hiệu và hành vi chức năng của 5 cổng cơ bản

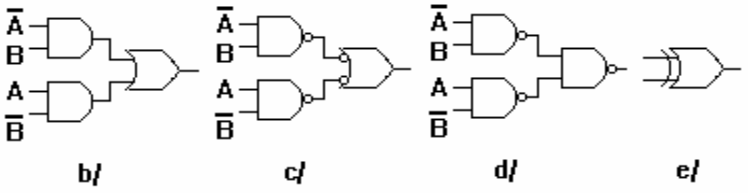
**Bài tập 1.2:** Tạo và mô phỏng hoạt động của các mạch logic số như trên hình 1.11 (Hình 3-03, GT KTMT), kiểm tra sự tương đương của chúng.




Hình 1.11 Các mạch logic có chức năng tương đương

**Bài tập 1.3:** Tạo và mô phỏng hoạt động của các mạch logic số như trên hình 1.12 (Hình 3-06, GT KTMT).

| A | B | XOR |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




Hình 1.12 Bảng chân lý của hàm XOR và các mạch điện thực hiện

# Bài số 2: Học sử dụng Logisim theo trợ giúp (tiếp)

## 2.1. Các yêu cầu và nội dung chính

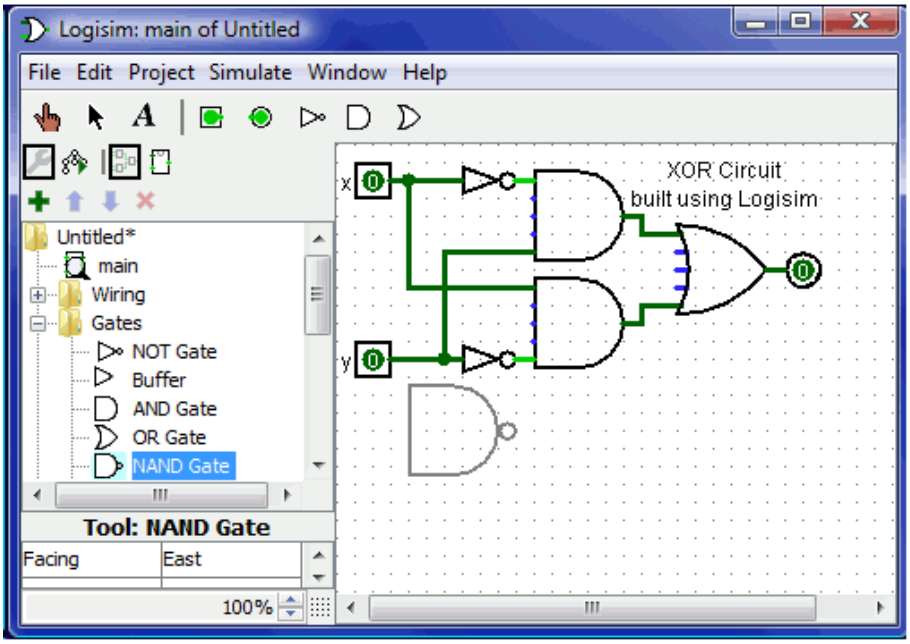
- Biết sử dụng thư viện của Logisim
- Biết bổ sung các mạch con (subcircuit) mà mình tạo ra vào thư viện
- Tạo và mô phỏng một số mạch điện có trong giáo trình Kiến trúc máy tính.

## 2.2. Sử dụng thư viện của Logisim và thay đổi thuộc tính của các phần tử logic

Thực hành trên máy theo "Help" của Logisim, xem mục **Libraries and Attributes**. (Help \ Tutorial \ Library and attributes). Các nội dung của mục 2.2 dưới đây được lược dịch từ mục nói trên. Trong mục này chúng ta tìm hiểu cách sử dụng hai vùng chính của cửa sổ Logisim, đó là Explorer pane (Ô cửa thăm dò) và Attribute table (Bảng thuộc tính).

### 2.2.1 Explorer pane (Ô cửa thăm dò)

Logisim tổ chức các công cụ của nó thành các thư viện. Mỗi nhóm công cụ được tổ chức như một folder, đặt trong Explorer pane. Chúng ta có thể mở folder bằng cách kích đúp vào biểu tượng của folder. Thí dụ, nhóm công cụ Gates gồm các công cụ (chúng ta có thể coi mỗi công cụ là một phần tử mạch điện) sau: NOT Gate, Buffer, AND Gate, OR Gate, NAND Gate, NOR Gate, XOR Gate, XNOR Gate …




Hình 2.1 Sử dụng công cụ "NAND Gate" để tạo ra phần tử NAND

Khi chúng ta muốn dùng một công cụ nào đó để tạo ra các phần tử mạch điện, thí dụ NAND Gate, hãy kích chuột vào biểu tượng của NAND Gate để chọn. Sau đó, nếu chúng ta rê chuột

sang cửa sổ Canvas, cứ mỗi lần kích chuột  thì có một phần tử NAND được đặt vào vị trí mà ta kích chuột.

Khi chúng ta bắt đầu tạo ra một mạch điện mới, chúng ta có thể sử dụng các nhóm công cụ có sẵn trong thư viện, gồm: Wiring, Gates, Plexers, Arithmetic, Memory, I/O và Base. Logisim cũng cho phép bổ sung thêm các công cụ vào thư viện có sẵn, bằng cách sử dụng thực đơn: Project \ Load Library \  Built-in librares | Logisim libraries | JAR libraries. Trong đó:

- Built-in librares: là các thư viện có sẵn trong bản Logisim được phân phát, được mô tả trong tài liệu kèm theo "Library Reference".
- Logisim libraries: Là các project mà người sử dụng xây dựng, được lưu lên đĩa và có thể đươc sử dụng như các mạch con (subcircuit).
- JAR libraries: Là các thư viện được người sử dụng phát triển bằng Javar nhưng không được phân phát cùng Logisim. Nếu người phát triển chia sẻ cho thì chúng ta có thể sử dụng (download về). Việc phát triển thư viện JAR khó hơn rất nhiều việc phát triển thư viện Logisim.

Chúng ta cũng có thể loại bỏ (remove) một số thư viện bằng cách sử dụng thực đơn: Project \ Unload libraries… Logisim không cho phép chúng ta loại bỏ thư viện có chứa các phần tử đã được sử dụng trong mạch điện và có xuất hiện trên Toolbar.

### 2.2.2 Attribute table (Bảng thuộc tính)

Nhiều phần tử mạch điện có thuộc tính (attributes) thay đổi được để người sử dụng có thể định cấu hình, làm cho nó có hành vi mong muốn. Thí dụ: phần tử D Flip-Flop có thuộc tính Trigger (kích hoạt), có thể thiết lập là "Rising Edge" hoặc "Falling Edge"…

Người sử dụng có thể dùng Edit tool kích vào phần tử (trên Canvas) mà mình muốn thay đổi thuộc tính rồi chọn một trong các giá trị cho thuộc tính trong Bảng thuộc tính. Nếu ta chọn đồng thời nhiều phần tử và muốn thiết lập thuộc tính cho chúng, thì trong bảng thuộc tính chỉ hiện các thuộc tính chung.

### 2.2.3 Tool attributes (Các thuộc tính của công cụ)

Các công cụ (Tool) dùng để tạo ra phần tử mạch điện cũng có một tập các thuộc tính, các thuộc tính này sẽ được truyền cho các phần tử tạo ra bởi công cụ.

Việc thay đổi thuộc tính của phần tử mạch điện (trên Canvas) không làm thay đổi thuộc tính của công cụ đã tạo ra nó (trên Explorer pane).

Cần chú ý:

- Việc thay đổi thuộc tính của một công cụ nào đó trên Toolbar không ảnh hưởng tới thuộc tính của chính công cụ đó ở Explorer pane, ngược lại cũng như vậy.
- Một số thuộc tính về định hướng phần tử mạch điện (facing) có thể thay đổi bằng cách gõ phím tắt (shortcut).

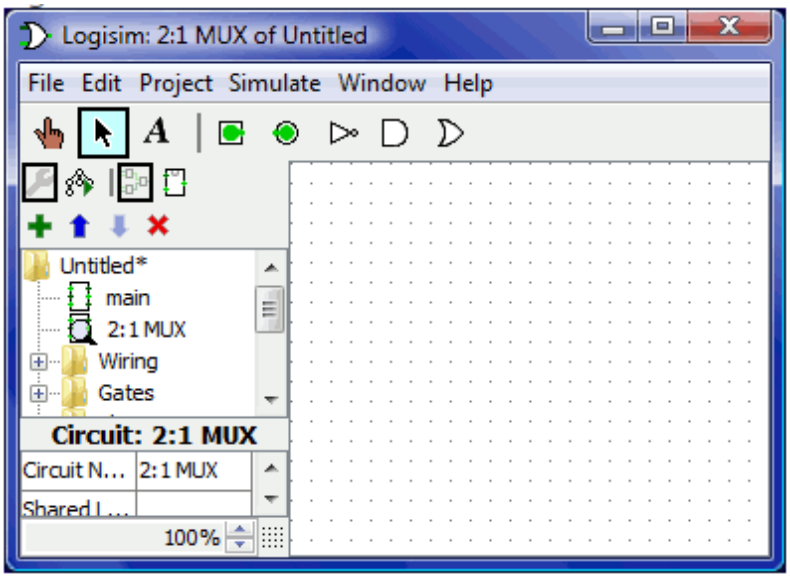
## 2.3. Tạo và sử dụng các mạch con

Thực hành trên máy theo "Help" của Logisim, xem mục **Subcircuits** (Help \ Tutorial \ Subcircuits). Các nội dung của mục 2.3 dưới đây được lược dịch từ mục nói trên.

Khi xây dựng các mạch điện ngày càng phức tạp chúng ta thường mong muốn xây dựng các mạch con đơn giản hơn, sử dụng chúng như những mô-đun để lắp ghép lại tạo thành mạch điện mà chúng ta cần xây dựng. Trong Logisim, các mạch con như vậy được gọi là subcircuit. Đối với người lập trình, subcircuit tương tự như subprogram, trong một số ngôn ngữ lập trình có thể được gọi là subroutine, function, method hay procedure. Mục đích của việc tạo ra các subcircuit cũng giống như việc người lập trình xây dựng chương trình từ các subprogram, đó là phân chia bài toán lớn thành nhiều bài toán nhỏ dễ thực hiện hơn và dễ tìm và sửa lỗi hơn.

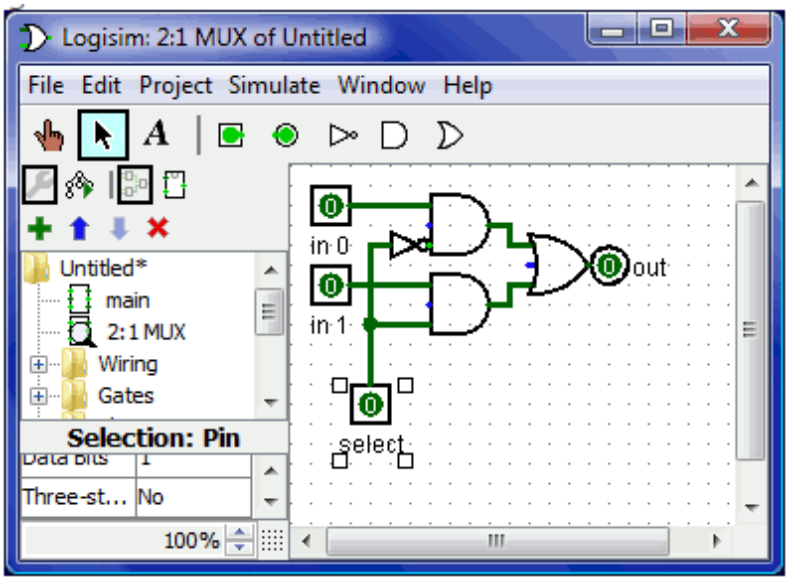
### 2.3.1 Tạo ra mạch con

Mỗi project trong Logisim thực tế là một thư viện các mạch điện. Project dạng đơn giản nhất chỉ gồm có một mạch điện, có tên là "Main" theo ngầm định (Default). Tuy nhiên, ta có thể dễ dàng bổ sung thêm subcircuit, chọn Project \ Add Circuit trên thanh thực đơn, sau đó đặt tên cho subcircuit. Thí dụ, nếu chúng ta muốn xây dựng một bộ dồn kênh 2-to-1 với tên là 2:1 MUX, sau khi bổ sung thêm mạch con theo cách nêu trên, cửa sổ Logisim nhìn thấy như trên hình 2.2.




Hình 2.2 Bổ sung mạch con "2:1 MUX"

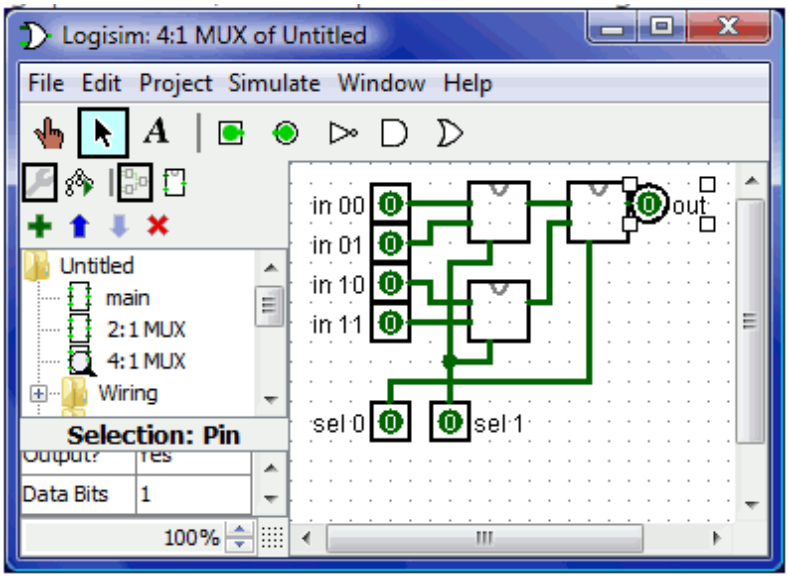
Ở cửa sổ Explorer pane, chúng ta sẽ nhìn thấy tên của hai mạch điện là "Main" và "2:1 MUX". Logisim hiện biểu tượng cái kính lúp bên trên biểu tượng của mạch con đang được hiện trên cửa sổ Canvas đồng thời tên của mạch con đó được hiển thị trên Title Bar (dòng trên cùng của cửa sổ). Bây giờ ta có thể xây dựng mạch con trên cửa sổ Canvas, khi kết thúc, "2:1 MUX" có thể thấy như ở hình vẽ 2.3 sau đây.




Hình 2.3 Tạo ra mạch con "2:1 MUX"

### 2.3.2 Sử dụng mạch con

Giả sử bây giờ chúng ta muốn xây dựng một bộ dồn kênh 4-to-1, ta có thể sử dụng mạch con 2-to-1 đã có của mình 3 lần. Hình ảnh của bộ dồn kênh 4-to-1 của chúng ta có thể như trên hình 2.4 dưới đây:




Hình 2.4 Sử dụng mạch con "2:1 MUX" để xây dựng mạch "4:1 MUX"

Sau khi đã xây dựng xong subcircuit 4:1 MUX, chúng ta có thể sử dụng nó để xây dựng các mạch con khác nếu muốn.

**Chú ý:** Tất nhiên là chúng ta có thể sửa subcircuit, nhưng cần nhớ rằng các thay đổi liên quan đến phần tử pin của subcircuit (bổ sung, xóa hoặc di chuyển) có thể sẽ làm thay đổi các mạch sử dụng subcircuit này.

### 2.3.3 Sửa mạch con

**Thể hiện của mạch con theo chế độ ngầm định**

Khi một subcircuit được đặt trong một mạch lớn hơn, subcircuit được Logisim biểu diễn bằng một hình chữ nhật với một đường khía (notch) chỉ thị hướng bắc của sơ đồ mạch điện. Các phần tử pin trong subcircuit được đặt trên cạnh của hình chữ nhật dựa trên hướng (facing) của nó trong subcircuit.

Ta có thể sử dụng tùy chọn thuộc tính "Shared Label" để gán nhãn cho subcircuit và làm cho nhãn này hiện trên mạch điện có sử dụng subcircuit. Nhãn này có thể thay đổi font và size như nhiều thuộc tính khác.

**Thể hiện của mạch con theo chế độ tùy chỉnh**

Thể hiện của mạch con theo chế độ ngầm định được sử dụng phổ biến từ khi Logisim ra mắt. Tuy nhiên, nếu muốn người sử dụng cũng có thể thay đổi được, hãy chọn Project \ Edit Circuit Appearance ở thanh thực đơn.
Đối với sinh viên học sử dụng Logisim để làm các bài tập môn Kiến trúc máy tính, chỉ nên sử dụng chế độ ngầm định.

### 2.3.4 Tìm và sửa lỗi (debugging) mạch con

Khi xây dựng các mạch điện thực hiện một chức năng yêu cầu nào đó, rất có thể mạch điện mà chúng ta xây dựng có lỗi (bug), nhất là khi mạch điện phức tạp, bao gồm nhiều subcircuit. Để xác định chính xác chỗ gây ra lỗi (hoặc sai), ta cần phải khảo sát tỉ mỉ từng mạch con tạo nên mạch điện của mình. Để vào xem trạng thái của một subcircuit, có thể sử dụng một trong 3 kỹ thuật sau:

- Đơn giản nhất là xem cấp bậc mô phỏng (simulation hierarchy) bằng cách kích vào biểu tượng thứ 2 ở Toolbar bên trên (biểu tượng cây trong tin học) hoặc là chọn "View Simulation Tree" ở thực đơn Project.
- Cách thứ hai là vào subcircuit để hiện Popup menu bằng cách kích nút chuột phải hoặc CTRL_click trên biểu tượng của subcircuit, sau đó chọn View option.
- Cách thứ ba là dùng Poke tool kích vào biểu tượng subcircuit mà ta muốn vào xem, trên subcircuit được chọn sẽ hiện lên biểu tượng cái kính lúp, nếu ta kích đúp lên cái kính lúp đó, thì sẽ vào xem được trạng thái của subcircuit.

Trong khi đang vào xem trạng thái của subcircuit, ta có thể thay đổi mạch điện này. Nếu các thay đổi của ta ảnh hưởng đến output của subcircuit thì các thay đổi này sẽ được truyền ra mạch điện sử dụng subcircuit. Có một ngoại lệ cần lưu ý là: các input của subcircuit được xác định dựa trên các giá trị đưa vào từ mạch mẹ (mạch sử dụng subcircuit), vì vậy việc dùng Poke tool để thay đổi giá trị các input này là không có ý nghĩa gì (Logisim sẽ nhắc nhở nếu chúng ta làm việc này).

**Chú ý:** Việc tìm và sửa lỗi (debugging) nói chung là tương đối khó, sinh viên sẽ tích lũy dần kinh nghiệm khi thực hiện các bài mô phỏng từ dễ đến khó theo tài liệu này.

### 2.3.5 Các thư viện của Logisim

Mỗi project của Logisim tự động là một thư viện có thể được nạp vào (load) các project khác. Tất cả các mạch điện (kể cả subcircuit) được định nghĩa ở một project nạp vào trước sẽ được Logisim coi như subcircuit cho project nạp vào tiếp theo. Đặc tính này của Logisim cho phép chúng ta tái sử dụng các thành phần dùng chung cho nhiều project và có thể chia sẻ với người sử dụng khác. Vì mỗi project đều có một Main circuit, nên Logisim cho phép chúng ta chọn một Main circuit làm Main circuit của project "mẹ".

Khi chúng ta xây dựng một project có tải vào các file project khác, ta có thể sử dụng các subcircuit của chúng nhưng không được phép sửa đổi mạch điện hoặc dữ liệu chứa trong đó (thí dụ dữ liệu gắn với các phần tử Constant, ROM…) Nếu chúng ta cần sửa, hãy mở các file project đó như một project độc lập, tiến hành sửa rồi lưu (save); sau đó có thể nạp lại vào (load).

## 2.4. Tra cứu theo các menu

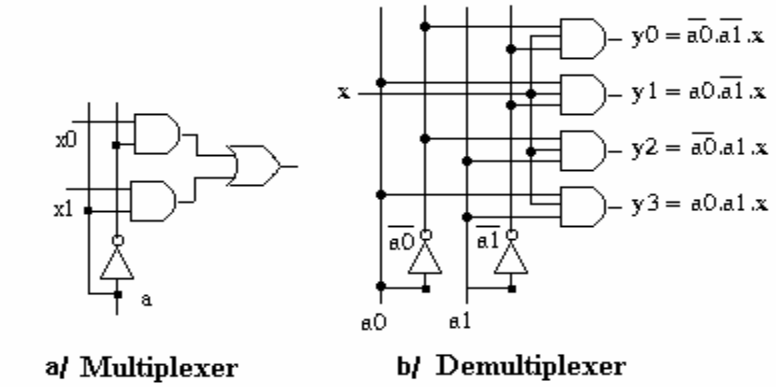
Thực hành trên máy theo "Help" của Logisim, xem mục **Menu Reference** (Help \ Tutorial \ Menu reference). Tại đây có mô tả về tất cả các phần tử có trong thư viện của Logisim. Trong quá trình học theo tài liệu này, sinh viên hãy tra cứu khi thấy cần thiết.

## 2.5. Bài tập (3 bài)

**Các bài tập này theo sát nội dung giáo trình Kiến trúc máy tính; Khi làm cần chú ý:**

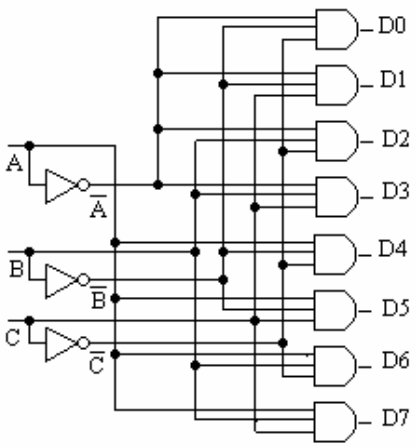
- Các mạch logic số mà sinh viên cần xây dựng trong các bài tập sau đây đã có trong thư viện của Logisim, nhưng sinh viên vẫn cần tạo ra chúng bằng các phần tử logic cơ bản để hiểu sâu về sự hoạt động của chúng.
- Sinh viên nên lấy từ thư viện một phần tử có chức năng tương tự chức năng của mạch logic số mà mình vừa tạo ra để so sánh chức năng của chúng.
- Sau này, khi xây dựng "Micro Architecture" (hình 4-07, hình 4-09 trong GT KTMT), chúng ta sẽ sử dụng các phần tử đã có trong thư viện.

**Bài tập 2.1:** Tạo và mô phỏng hoạt động của multiplexer, demultiplexer trên hình 2.5 (Hình 3-07, GT KTMT).



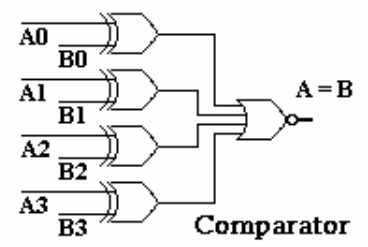

Hình 2.5 Bộ dồn kênh và bộ phân kênh

**Bài tập 2.2:** Tạo và mô phỏng hoạt động của bộ decoder "3 to 8" trên hình 2.6 (Hình 3-08, GT KTMT).




Hình 2.6 Mạch giải mã "3 to 8"

**Bài tập 2.3:** Tạo và mô phỏng hoạt động của bộ so sánh 4 bit trên hình 2.7 (Hình 3-09, GT KTMT).




Hình 2.7 Bộ so sánh 4 bit

# Bài số 3: Học sử dụng Logisim theo trợ giúp (tiếp)

## 3.1. Các yêu cầu và nội dung chính

- Biết tạo và sử dụng các bó dây (wire bundles)
- Biết phân tích, tạo và mô phỏng các mạch tổ hợp (combinational circuit)
- Tạo và mô phỏng được các mạch tổ hợp có trong giáo trình Kiến trúc máy tính.

## 3.2. Tạo và sử dụng các bó dây

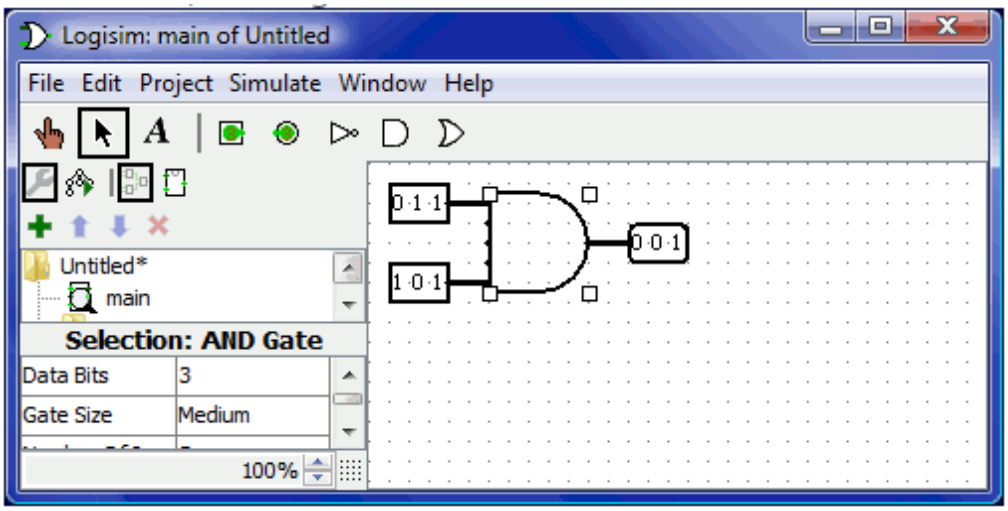
Thực hành trên máy theo "Help" của Logisim, xem mục **Wire bundles** (Help \ Tutorial \ Wire bundles). Các nội dung của mục 3.2 dưới đây được lược dịch từ mục nói trên.

Trong cách mạch điện đơn giản, hầu hết các dây điện chỉ truyền 1 bit; Tuy nhiên, Logisim cũng cho phép chúng ta tạo ra một bó dây truyền song song nhiều bit, bó dây này được thể hiện như một dây trên "bản vẽ" mạch điện, làm cho mạch điện trông đơn giản hơn.

### 3.2.1 Tạo ra bó dây

Mỗi input và output của mọi phần tử mạch điện đều có độ rộng bit (bit width) nhất định. Thường thường độ rộng bit là 1; tuy nhiên nhiều phần tử trong thư viện của Logisim có thuộc tính Bit width thay đổi được từ 1 đến tối đa là 32.

Hình vẽ dưới đây là một thí dụ, trong đó cổng AND có 5 input (nhưng chỉ có 2/5 input có dữ liệu đưa vào), bit width của mỗi input là 3. Đầu ra (phần tử pin với thuộc tính "output") cũng là 3 bit, trong đó mỗi bit là AND của 2 bit cùng bậc của 2 input đưa vào.




Hình 3-1 Bó dây nối cổng AND với các phần tử input và ouput

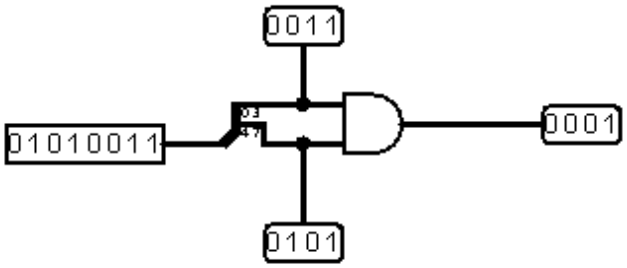
Như vậy để truyền các input vào và ra phần tử AND ta phải dùng các bó dây, mỗi bó có 3 "sợi". Khi chúng ta nối dây với một đầu vào hay đầu ra của một phần tử nào đó thì Logisim tự động chọn Bit width của dây bằng Bit width của đầu vào hay đầu ra của phần tử được nối với dây. Nếu Bit width của 2 phần tử mạch điện ở 2 đầu dây khác nhau, Logisim sẽ báo lỗi ngay trên mạch điện: "Incompatible width".

Nếu dây nối chỉ gồm 1 sợi dây, thì khi truyền bit 1 hay 0 Logisim sẽ hiện dây đó với màu xanh lá cây nhạt (light green) hoặc sẫm (dark green). Nếu dây nối là một bó dây nhiều sợi, thì Logisim chỉ sử dụng màu đen.

### 3.2.2 Công cụ tách/gộp bó dây - Splitters

Khi chúng ta làm việc với các giá trị nhiều bit, chúng ta có thể cần truyền các nhóm bit khác nhau trong số đó đến các đích khác nhau. Công cụ Splitter có sẵn trong thư viện của Logisim có chức năng giúp chúng ta thực hiện việc này.

Thí dụ, giả sử chúng ta cần xây dựng mạch điện thực hiện AND các cặp bit tương ứng của 2 nửa byte (nibble) của input 8 bit. Khi đó chúng ta sẽ cần tách 4 bit thấp và 4 bit cao của 8 bit đầu vào, sau đó đưa 2 nhóm 4 bit vào 2 đầu vào của một cổng AND. Mạch điện có sử dụng phần tử Splitter như ở hình dưới đây.




Hình 3-2 Minh họa việc sử dụng một phần tử Splitters

Trong mạch điện này, phần tử Splitter thực hiện việc tách các nhóm dây (tách 8 dây thành 2 nhóm 4 dây) và có các thuộc tính như sau:

- Facing: East
- Fan Out: 2
- Bit Width: 8
- Appearance: Left-handed
- ...

Tất nhiên, Splitter cũng có thể gộp các nhóm dây lại thành một bó dây. Thực chất, phần tử Splitter truyền theo cả hai hướng như nhau.

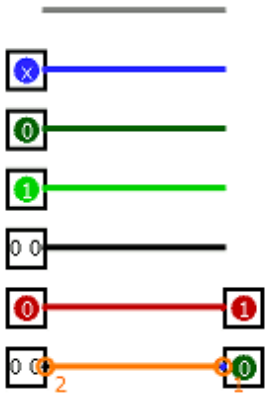
### 3.2.3 Các mầu của dây

Phần tử dây (Wire) có thể có các màu như ở hình 3-3 dưới đây:

- Gray (màu xám, trên cùng): độ rộng bit (bit width) của dây là không biết. Điều này có thể xảy ra khi dây không nối với một phần tử input hay output nào.
- Blue (xanh da trời): Dây đang truyền bit 1, nhưng không có phần tử mạch điện nào đưa giá trị đó lên dây, nói cách khác là dây đó được thả nổi – floating. Trạng thái này cũng

còn được gọi là có giá trị trở kháng cao (high-impedance value). Phần tử 3 trạng thái có thể đặt giá trị này lên dây.

- Dark green (xanh lá cây xẫm): Dây đang truyền bit 0.
- Bright green (xanh lá cây nhạt): Dây đang truyền bit 1.
- Black (đen): Dây đang truyền một giá trị nhiều bit.
- Red (đỏ): Dây đang truyền một giá trị sai, thường là do có một cổng không thể xác định được giá trị ra - output thích hợp (đúng đắn), có thể là do không có input. Trạng thái này cũng có thể là do có 2 phần tử cùng truyền các giá trị khác nhau lên dây, gây nên đụng độ (conflict). Một bó dây cũng có thể có màu đỏ nếu có một sợi dây truyền giá trị sai.
- Orange (Da cam): Phần tử nối với dây không có cùng độ rộng bit (bit width) với dây.



Hình 3-3 Các màu có thể có của phần tử dây (wire) trong mạch điện

## 3.3. Phân tích các mạch tổ hợp (Combinational circuits)

Việc phân tích mạch tổ hợp không thuộc nội dung thực hành được viết trong tài liệu này, vì vậy mục 3.3 này chỉ trình bày sơ lược để sinh viên tham khảo khi tự học.

Thực hành trên máy theo "Help" của Logisim, xem mục **Combinational analysis**. (Help \ Tutorial \ Combinational analysis)

Khi học chương 3 của giáo trình Kiến trúc máy tính chúng ta đã biết rằng mạch tổ hợp (combinational circuit) là loại mạch điện mà giá trị ở các đầu ra là hàm lô-gic của các giá trị đầu vào ở thời điểm ta xét. Để mô tả hành vi (behavior) của mạch tổ hợp, người ta thường sử dụng ba kỹ thuật chính:

- Sử dụng mạch logic
- Sử dụng biểu thức logic, kỹ thuật này cho phép biểu diễn dưới dạng đại số sự làm việc của mạch điện.
- Sử dụng bảng chân lý (Truth table) liệt kê tất cả các kết hợp đầu vào có thể có và các giá trị ra tương ứng.

Mô đun chức năng "Combinational Analysis" của Logisim cho phép chúng ta chuyển đổi giữa ba cách mô tả hành vi nêu trên. Chức năng này đặc biệt thuận tiện cho chúng ta khi cần tạo ra và/hoặc tìm hiểu các mạch điện có rất nhiều input và output kiểu 1 bit.

**Mở cửa sổ "Combinational Analysis"**

Logisim có nhiều mô-đun có chức năng phân tích mạch tổ hợp, tất cả có thể truy cập được trong một cửa sổ có tên là "Combinational Analysis". Có thể mở cửa sổ này theo hai cách được trình bày dưới đây.

**Mở từ mục chọn "Window" trên thực đơn**

Chọn mục "Window", sau đó chọn mục con "Combinational Analysis" thì cửa sổ đó sẽ hiện ra. Nếu trước đây chúng ta chưa từng mở cửa sổ này, thì cửa sổ mới mở không chứa mạch điện nào.

Khi chạy Logisim, dù chúng ta có mở đồng thời nhiều project, nhưng Logisim chỉ có một cửa sổ "Combinational Analysis".

**Mở từ mục chọn "Project" trên thực đơn**

Chọn mục "Project", sau đó chọn mục con "Analyze Circuit" thì cửa sổ đó sẽ hiện ra. Trước khi hiện cửa sổ, Logisim tính các biểu thức logic và bảng chân lý liên quan đến mạch điện và hiện ở cửa sổ này cho người sử dụng thấy.

Để Logisim có thể tính được, tất cả các đầu vào của các phần tử mạch điện phải được nối với phần tử pin với thuộc tính input. Logisim cũng chỉ có khả năng phân tích mạch điện có tối đa 8 phần tử pin như vậy, mỗi phần tử có bit width là 1.

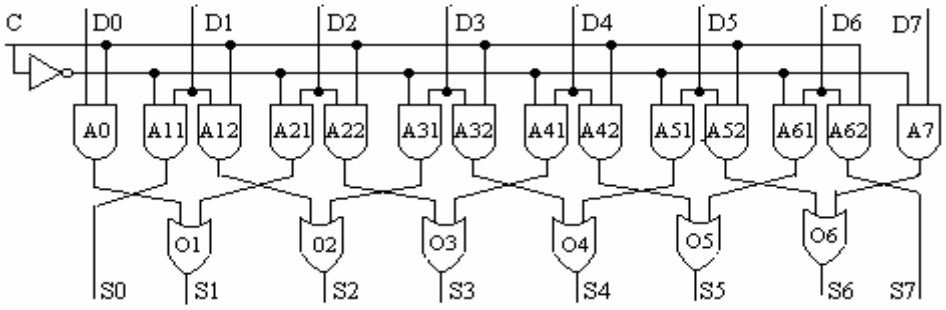
Nếu các điều kiện như trên không đúng, Logisim sẽ có thông báo lỗi và không mở cửa sổ "Analyze Circuit".

## 3.4. Bài tập (4 bài)

**Các bài tập này theo sát nội dung giáo trình Kiến trúc máy tính; Khi làm cần chú ý:**

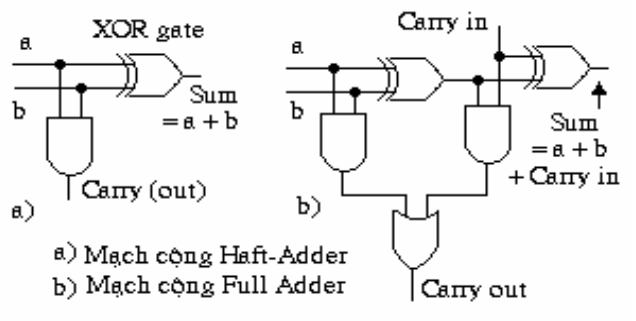
- Một số mạch logic số sau đây đã có trong thư viện của Logisim, nhưng sinh viên vẫn cần tạo ra chúng bằng các phần tử logic cơ bản để hiểu sâu về sự hoạt động của chúng.
- Sinh viên nên lấy từ thư viện một phần tử có chức năng tương tự chức năng của mạch logic số mà mình vừa tạo ra để so sánh chức năng của chúng.
- Sau này, khi xây dựng "Micro Architecture" (hình 4-07, hình 4-09), chúng ta sẽ sử dụng các phần tử đã có trong thư viện.

**Bài tập 3.1:** Tạo và mô phỏng hoạt động của bộ dịch (shifter) trên Hình 3-4 (Hình 3-11, GT KTMT). Thực tập sử dụng các phần tử "Shifter" (thuộc nhóm "Arithmetic") và "Shift Register" (thuộc nhóm "Memory") có trong thư viện của Logisim (trong các nhóm "Arithmetic" và "Memory").



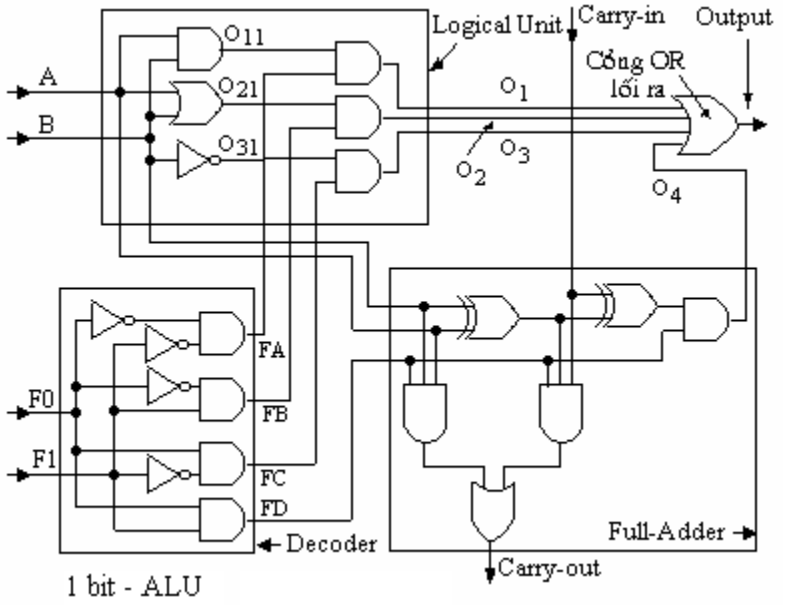

Hình 3.4 Bộ dịch 1 bit sang trái/phải

**Bài tập 3.2:** Tạo và mô phỏng hoạt động của các bộ cộng Haft-Adder và Full-Adder 1 bit theo sơ đồ trên Hình 3-5 (Hình 3-12, GT KTMT). Thực tập sử dụng phần tử "Adder" có trong thư viện của Logisim (thuộc nhóm "Arithmetic").



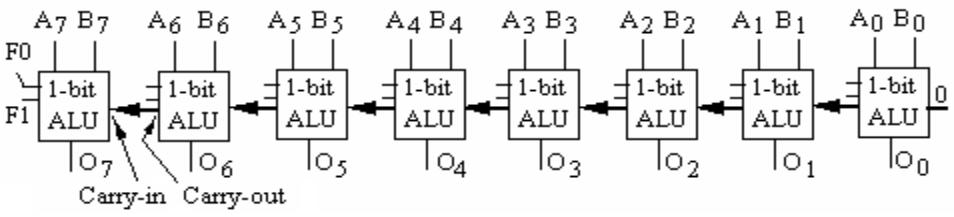

Hình 3.5 Bộ cộng

**Bài tập 3.3:** Tạo và mô phỏng hoạt động của bộ số học là logic trên Hình 3-6 (1-bit ALU) (Hình 3-13, GT KTMT). Lưu (save) vào file "ALU-1bit.circ" để sử dụng cho bài tập sau.




Hình 3.6 Bộ Số học và Logic (ALU) 1 bit

**Bài tập 3.4:** Tạo và mô phỏng hoạt động của ALU 8 bit từ các mạch con là ALU 1 bit như trên hình 3-7 (sử dụng file ALU-1bit.circ) (Hình 3-14, GT KTMT).




Hình 3.7 ALU 8-bit được tạo bởi "8 mảnh" ALU 1-bit

# Bài số 4: Học sử dụng Logisim theo trợ giúp (tiếp)

## 4.1. Các yêu cầu và nội dung chính

- Biết được khả năng mô phỏng bộ nhớ của Logisim.
- Biết tạo và mô phỏng sự hoạt động của các phần tử nhớ cơ bản, như: D Flip-Flop, S-R Flip-Flop, Register.

## 4.2. Các phần tử bộ nhớ

Thực hành trên máy theo "Help" của Logisim, xem mục **Memory components** (Help \ Tutorial \ Memory components) và tra cứu thêm ở mục **Memory library** (Help \ Tutorial \ Library Reference \ Memory Library). Các nội dung của mục 4.2 này được lược dịch từ các mục trên.

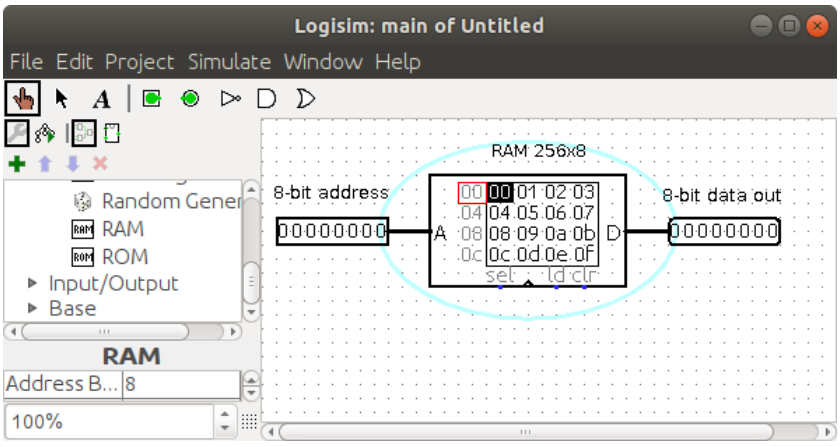
RAM và ROM là hai phần tử bộ nhớ hay được sử dụng nhất trong thư viện của Logisim, đồng thời chúng cũng là các phần tử nhớ phức tạp nhất. Các mục con dưới đây chủ yếu hướng dẫn cách xem và sửa nội dung các phần tử nhớ này.

### 4.2.1 Thay đổi nội dung bộ nhớ (Poking memory)

Chúng ta có thể sửa bằng tay nội dung của phần tử nhớ bằng Poke tool; Tuy Poke tool có một giao diện rất đơn giản, nhưng có khả năng soạn thảo số Hexa rất thuận lợi.

Poke tool có 2 chế độ hoạt động để xem và sửa nội dung phần tử nhớ (cả RAM và ROM): Sửa địa chỉ của miền ô nhớ được hiển thị và sửa nội dung từng ô nhớ.

Để sửa địa chỉ của miền ô nhớ được hiển thị, ta dùng Poke tool kích lên một vị trí trên cột ghi địa chỉ (số Hexa) nằm bên ngoài hình chữ nhật chứa nội dung các ô nhớ, Logisim sẽ hiện một hình chữ nhật màu đỏ bao quanh con số (địa chỉ) trên cùng.

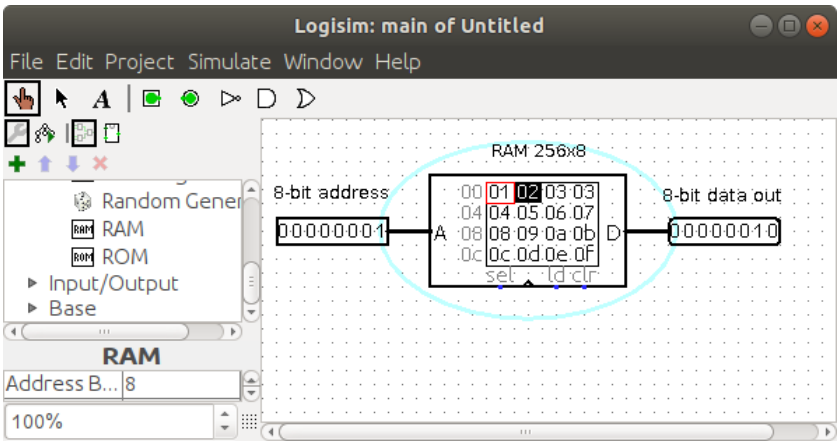



Hình 4-1 Thí dụ về sửa địa chỉ của miền ô nhớ RAM được hiển thị

- Bây giờ ta có thể sửa con số này, bằng cách nhập từ bàn phím con số mà chúng ta muốn ghi đè lên con số bên trong hình chữ nhật màu đỏ. Con số mà chúng ta nhập vào sẽ là địa chỉ của ô nhớ đầu tiên của miền ô nhớ được logisim hiển thị trên cửa sổ nhỏ. Các ô nhớ có địa chỉ nhỏ hơn sẽ không thấy được, trừ khi chúng ta truyền vào đầu vào địa chỉ giá trị địa chỉ mới nhỏ hơn con số chúng ta vừa nhập vào.

- Khi nhấn Enter Logisim sẽ chọn con số ở dòng tiếp theo bên dưới.
- Khi nhấn phím Backspace Logisim  sẽ chọn con số ở dòng bên trên.
- Khi nhấn phím Space Logisim  sẽ cuộn xuống một trang.

Để sửa giá trị (con số Hexa) một ô nhớ cụ thể, ta dùng Poke tool kích lên con số đó nằm bên trong hình chữ nhật chứa nội dung các ô nhớ, Logisim sẽ hiện một hình chữ nhật màu đỏ bao quanh con số đó.




Hình 4-2 Thí dụ về sửa nội dung của  một ô nhớ RAM

- Bây giờ ta có thể sửa con số này, bằng cách nhập từ bàn phím con số mà chúng ta muốn ghi đè lên con số bên trong hình chữ nhật màu đỏ.
- Khi nhấn Enter Logisim sẽ chọn con số ở dòng tiếp theo bên dưới.
- Khi nhấn phím Backspace Logisim sẽ chọn con số của ô nhớ ở địa chỉ trước.
- Khi nhấn phím Space Logisim sẽ chọn con số của ô nhớ ở địa chỉ sau.

### 4.2.2 Các thực đơn pop-up và file (Pop-up menus and files)

Thực đơn pop-up cho bộ nhớ RAM và ROM, ngoài các tùy chọn chung cho các phần tử nhớ khác còn có 4 tùy chọn:

- Edit Contents (Sửa nội dung): Gọi trình biên soạn số Hex để sửa nội dung bộ nhớ.
- Clear Contents (Xóa nội dung): Xóa (reset) tất cả nội dung trong bộ nhớ.
- Load Image (Nạp hình ảnh): Nạp nội dung bộ nhớ đã lưu trữ dưới dạng file vào phần tử RAM/ROM được chọn; File này có định dạng (format) được mô tả dưới đây.
- Save Image (Lưu hình ảnh): Chứa (store) nội dung bộ nhớ (RAM/ROM) vào file; File này có định dạng (format) được mô tả dưới đây.

Định dạng của file chứa hình ảnh bộ nhớ: File này chỉ chứa số Hexa thuộc bộ ký tự ASCII chuẩn, trừ dòng đầu tiên có nội dung là: v2.0 raw. Thí dụ, nếu ta có bộ nhớ kích thước 256 byte, trong đó 5 byte đầu tiên là 2, 3, 0, 20 và -1, còn các giá trị tiếp theo (251 byte) đều là 0, thì hình ảnh bộ nhớ được lưu trong file sẽ thấy như sau:

v2.0 raw

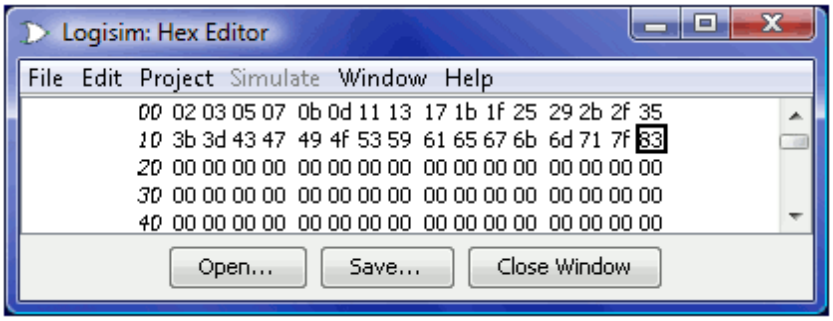
02 03 00 14 ff 00 … 00

Dòng đầu tiên (“v2.0 raw”) chỉ ra định danh (id) của format; Tuy nhiên cho đến nay - năm 2020, Logisim chỉ có một loại định danh này. Các số tiếp theo là số Hexadecimal. Chú ý: ff chính là dạng nhị phân âm bù 2 của -1 thập phân.

Một số chi tiết nữa về định dạng file có thể tra cứu được trong mục trợ giúp của Logisim (Help).

### 4.2.3 Trình soạn thảo số Hex (Hex editor)

Logisim có bên trong nó một trình soạn thảo số Hex để xem và sửa nội dung bộ nhớ. Để gọi trình soạn thảo này ra làm việc, chỉ cần sử dụng pop-up menu như đã trình bày ở mục con bên trên (mục 4.2.2). Ngoài ra, với ROM cũng còn một cách khác nữa, đó là chọn phần tử nhớ rồi kích vào dòng “Contents” ở bảng thuộc tính (trong cửa sổ con ở góc dưới bên trái cửa sổ Logisim). Hình 4.1 dưới đây là thí dụ về một cửa sổ được mở ra khi chúng ta kích hoạt trình soạn thảo “Hex editor”.




Hình 4-3 Thí dụ về cửa sổ Hex editor

Các số in nghiêng ở cột bên trái là địa chỉ ô nhớ, các số còn lại là nội dung các ô nhớ. Hex editor có thể hiện trên mỗi dòng 4, 8 hoặc 16 số Hex 2 chữ/số, tùy theo việc chúng ta mở rộng cửa sổ trình soạn thảo đến mức độ nào. Với Hex editor chúng ta có thể thực hiện các thao tác soạn thảo quen thuộc, như: Cut, Copy, Paste, Delete, Select All… bằng cách sử dụng thực đơn Edit của nó.

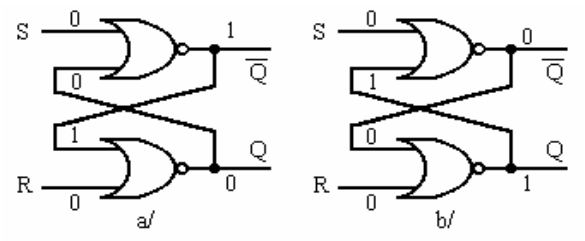
Một số chi tiết nữa về việc sử dụng trình soạn thảo Hex editor có thể tra cứu được trong mục trợ giúp của Logisim (Help).

## 4.3. Bài tập (4 bài)

**Các bài tập này theo sát nội dung giáo trình Kiến trúc máy tính; Khi làm cần chú ý:**

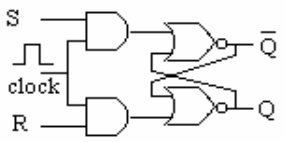
- Một số mạch logic số sau đây đã có trong thư viện của Logisim, nhưng sinh viên vẫn cần tạo ra chúng bằng các phần tử logic cơ bản để hiểu sâu về sự hoạt động của chúng.
- Sinh viên nên lấy từ thư viện một phần tử có chức năng tương tự chức năng của mạch logic số mà mình vừa tạo ra để so sánh chức năng của chúng.
- Sau này, khi xây dựng “Micro Architecture” (hình 4-07, hình 4-09), chúng ta sẽ sử dụng các phần tử đã có trong thư viện.

**Bài tập 4.1:** Tạo và mô phỏng sự hoạt động của thanh ghi chốt đơn giản nhất - thanh ghi chốt RS, theo Hình 4.4 (Hình 3-16 trong GT KTMT); sau đó lặp lại việc này nhưng sử dụng các phần tử NAND thay cho các phần tử NOR.



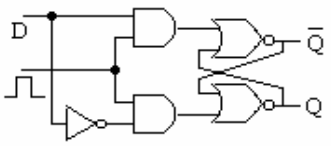

Hình 4.4 a/ Thanh ghi chốt dùng cổng NOR ở trạng thái 0; b/ Thanh ghi chốt dùng cổng NOR ở trạng thái 1

**Bài tập 4.2:** Tạo và mô phỏng sự hoạt động của thanh ghi chốt RS hoạt động theo nhịp xung đồng hồ theo Hình 4.5 (Hình 3-17 trong GT KTMT). Chú ý thử tín hiệu đồng hồ (clock) theo 2 cách: 1/ sử dụng phần tử "input" và dùng phần tử "Poke tool" để thay đổi giá trị của "input"; 2/ sử dụng đơn vị tạo tín hiệu clock thuộc nhóm "Base" trong thư viện của Logisim.




Hình 4.5 Thanh ghi chốt RS hoạt động theo nhịp xung đồng hồ

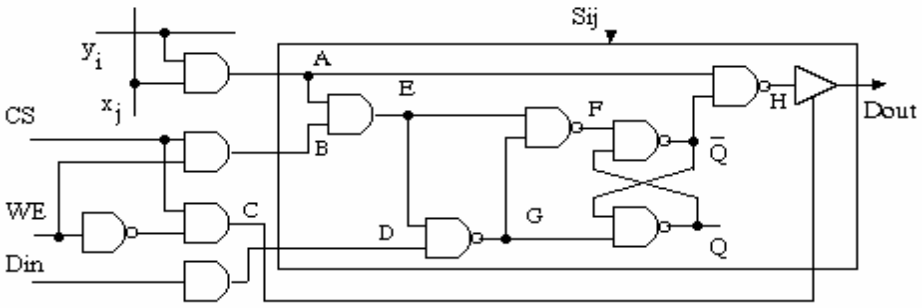
**Bài tập 4.3:** Tạo và mô phỏng sự hoạt động của thanh ghi chốt D hoạt động theo nhịp xung đồng hồ như trên hình 4.6 (Hình 3-18, GT KTMT).




Hình 4.6 Thanh ghi chốt D hoạt động theo nhịp xung đồng hồ

**Bài tập 4.4:** Tạo và mô phỏng sự hoạt động của phần tử nhớ 1 bit của bộ nhớ RAM tĩnh (SRAM) như trên hình 4.7 (Hình 3-19, GT KTMT).

Chú ý: (1) Sử dụng phần tử "Controlled Buffer" thuộc nhóm "Gates" trong thư viện làm bộ đệm 3 trạng thái có đầu ra nối với "Dout". (2) Sử dụng phần tử "Buffer" thuộc nhóm "Gates" trong thư viện làm bộ đệm có đầu vào nối với "Din".

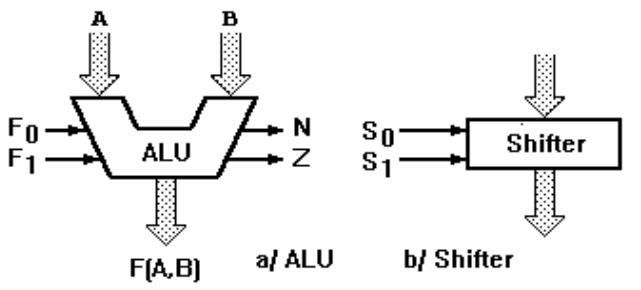



Hình 4.7 Mạch điện của một phần tử nhớ RAM tĩnh 1 bit Sij

# Bài số 5: Xây dựng ALU-16 bit của đường dữ liệu

## 5.1. Các yêu cầu và nội dung chính

Xây dựng và mô phỏng sự hoạt động của đơn vị ALU của vi kiến trúc được mô tả trên hình 5.1. ALU này được mô tả chi tiết tại mục "4.1.4 Đơn vị Số học-Logic và Bộ dịch" và được minh hoạ trên hình 4-04 của GT KTMT.




Hình 5.1 a/ Ký hiệu một ALU; b/ Ký hiệu một Shifter (Hình 4-04 trong GT KTMT)

ALU này thực hiện một trong 4 phép tính trên tóan hạng 16 bit, theo mã phép tóan chứa trong trường ALU (gồm 2 bit F0 và F1) của thanh ghi MIR chứa vi chỉ thị đang thi hành:

- ALU (F1 F0) = 00: Cộng 2 tóan hạng 16 bits ở 2 đầu vào, số nhớ đưa vào (carry-in) bằng 0, số nhớ đưa ra được sử dụng để tạo thành các tín hiệu trạng thái N và Z.
- ALU (F1 F0) = 01: AND 2 tóan hạng 16 bits ở 2 đầu vào.
- ALU (F1 F0) = 10: Cho tóan hạng ở đầu vào bên trái (đầu vào A) đi ra đầu ra, không thay đổi; các bít trạng thái N và Z được thiết lập phụ thuộc vào kết quả ở đầu ra.
- ALU (F1 F0) = 11: Đảo tóan hạng ở đầu vào bên trái (đầu vào A) và đưa ra đầu ra.

## 5.2. Phân tích bài tóan

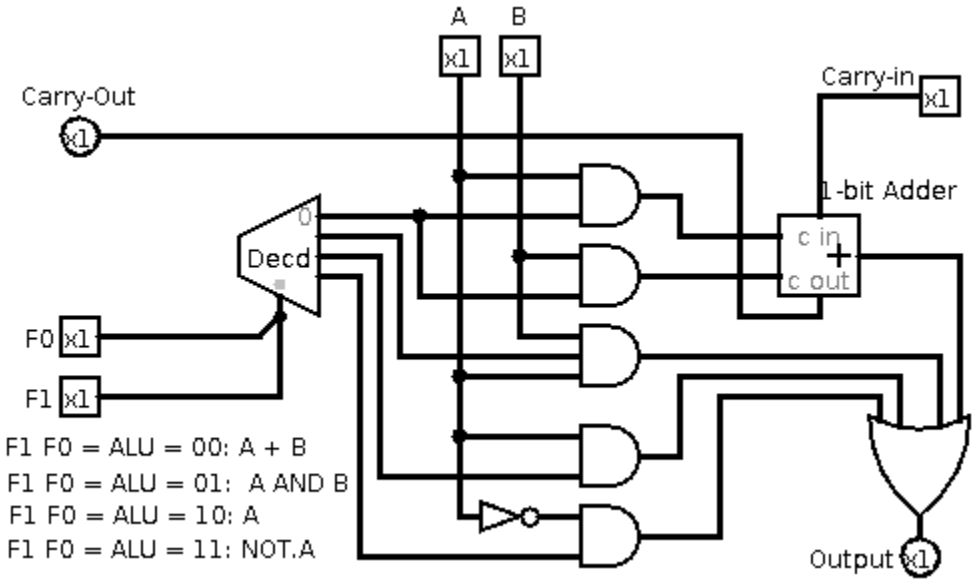
- ALU này thực hiện các chức năng không hoàn toàn giống với ALU đã được trình bày trong GT KTMT (mục 3.2.3.3 Bộ số học và logic - ALU). Tuy nhiên, sự khác nhau là không nhiều. Chúng ta chỉ phải tạo ra một mạch điện thực sự mới để sinh ra tín hiệu trạng thái N và Z, theo nguyên tắc sau:
  - N - chỉ báo rằng kết quả ra của ALU là âm (Negative). Thực chất, bit N chỉ là copy của bit có bậc cao nhất của kết quả đưa ra.
  - Z - chỉ báo rằng kết quả ra của ALU bằng không (Zero). Thực chất, bit Z là NOR của tất cả các bit của kết quả đưa ra.
  - Khi xây dựng ALU-16 bits:
    + Carry-in cho bộ cộng 2 bit bậc 0 được cho bằng 0.
    + Carry-out của bộ cộng 2 bit bậc cao nhất (15) không cần quan tâm (mục đích là để mạch điện và việc phân tích nó không quá phức tạp).

- Để việc xây dựng mạch điện theo kiểu "mô-đun" hóa, chúng ta sẽ xây dựng mạch ALU 1 bit trước, lưu vào file ALU-1bit.circ; sau đó sử dụng nó để xây dựng ALU-8 bits; cuối cùng xây dựng ALU-16 bits theo yêu cầu từ 2 ALU 8 bit và một số phần tử mạch điện khác, thí dụ mạch tạo các tín hiệu trạng thái N và Z.

**Chú ý:** Chúng ta có thể xây dựng 1 ALU-16 bits từ 16 ALU-1 bit, thậm chí từ các phần tử logic cơ bản. Tuy nhiên, nếu làm như vậy sẽ không thể nhìn thấy toàn bộ mạch điện trên màn hình.

## 5.3. Xây dựng ALU 1 bit

- Chạy Logisim; Vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là ALU-1bit, không cần đánh vào tên mở rộng luôn được Logisim đặt mặc định là ".circ". Trong cửa sổ con ở góc trên bên trái, ngoài tên main, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là ALU-1bit.
- Lưu (save) mạch điện mà chúng ta tạo ra: vào menu File \ Save | Save As. Khi Logisim hỏi tên mà chúng ta muốn đặt, hãy đặt tên là: Micro-Architecture-SVxx.circ. Trong đó, SVxx là mã số sinh viên (hoặc số thứ tự của sinh viên trong danh sách) và ".circ" là tên mở rộng (mặc định). Trong thời gian thực hành, hoặc khi kết thúc, nên thường xuyên "save".
- Xây dựng ALU 1 bit như ở hình 5.2 dưới đây.




Hình 5.2 ALU 1 bit với các đầu vào, ra và điều khiển

**Chú ý:**

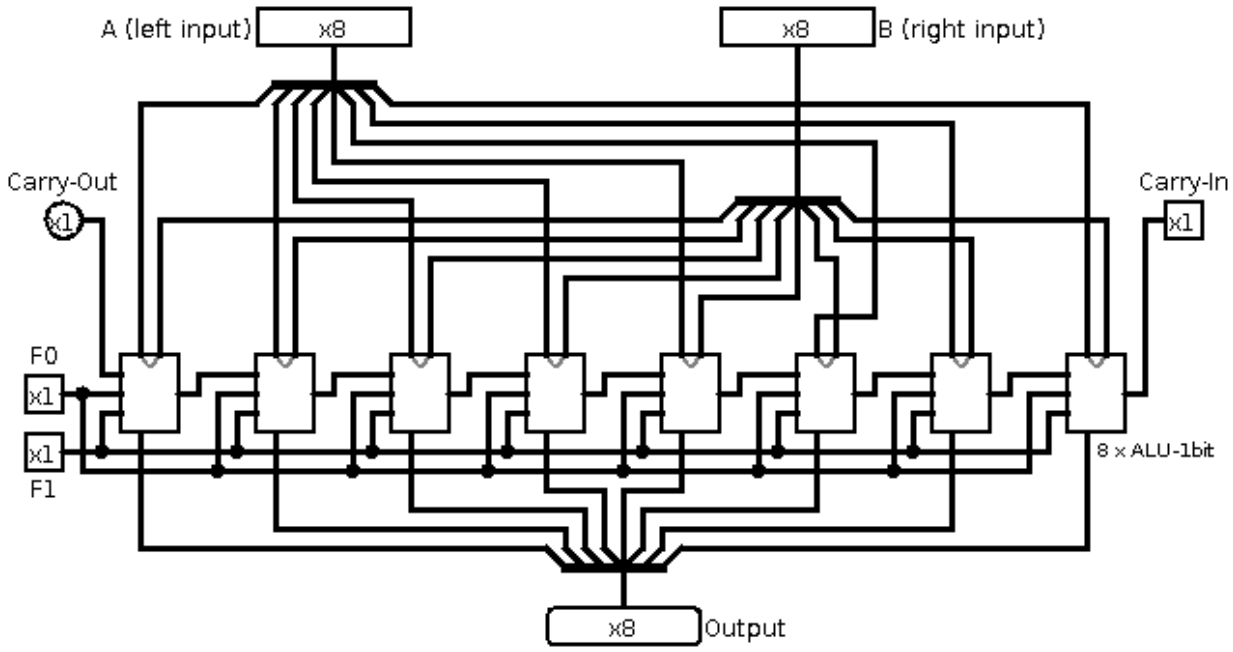
1. Mạch con ALU-1bit này sẽ được chúng ta sử dụng để xây dựng ALU-8 bit.
2. Các phần tử input/output của mạch con nằm ở hướng nào (North, South, East, West) của mạch điện này, thì khi chúng ta sử dụng mạch con này như một phần tử của thư viện, các đầu ("chân") input/ouput của phần tử đó sẽ có cùng hướng (North, South, East, West).

- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).

- Kiểm tra sự thực hiện 4 chức năng của ALU bằng cách dùng phần tử "Poke tool" để thiết lập các giá trị khác nhau của các đầu vào: F0, F1, A, B và Carry-in; đồng thời quan sát giá trị trên các đầu ra: Output và Carry-Out.

## 5.4. Xây dựng ALU 8 bit

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là ALU-8bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và ALU-1bit, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là ALU-8bits.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con ALU-8bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Xây dựng ALU-8bits như ở hình dưới đây. Chú ý: hãy sử dụng các phần tử ALU-1bit do chúng ta tạo ra. Việc này không khác gì so với việc sử dụng các phần tử có sẵn trong thư viện của Logisim.
- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).
- Kiểm tra sự thực hiện 4 chức năng của ALU bằng cách dùng phần tử "Poke tool" để thiết lập các giá trị khác nhau của các đầu vào: F0, F1, A, B và Carry-in; đồng thời quan sát giá trị trên các đầu ra: Output và Carry-Out.



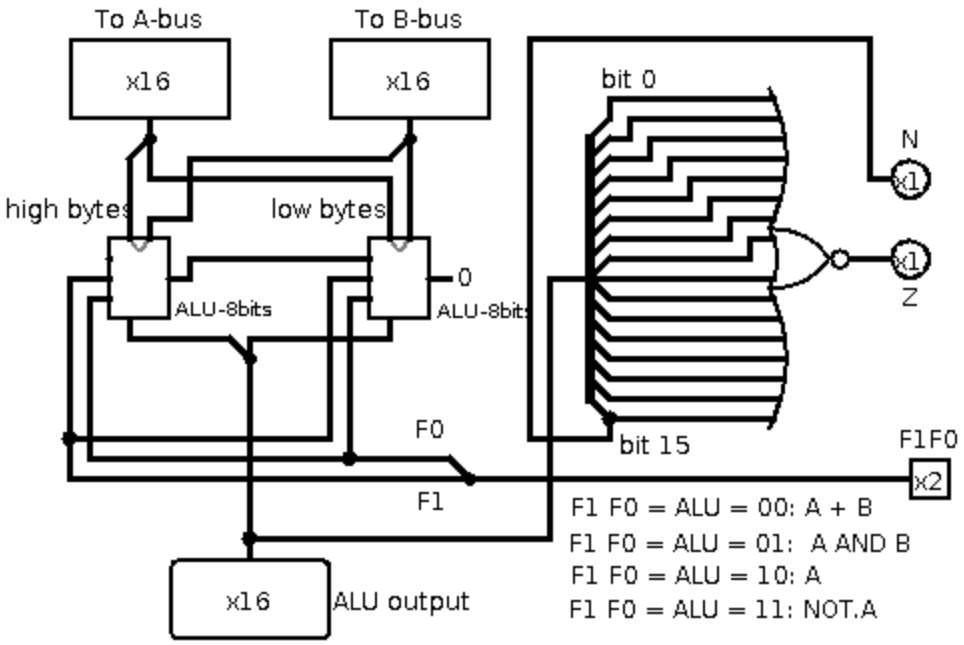

Hình 5.3 ALU 8 bit được xây dựng từ 8 ALU 1 bit

## 5.5. Xây dựng ALU 16 bit hoàn chỉnh

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là

ALU-16bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main, ALU-1bit, ALU-8bits, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là ALU-16bits.

- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con ALU-16 bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Xây dựng ALU-16 bits như ở hình 5.4 dưới đây. Chú ý: hãy sử dụng các phần tử ALU-8 bits do chúng ta tạo ra.
- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).
- Kiểm tra sự thực hiện 4 chức năng của ALU bằng cách dùng phần tử "Poke tool" để thiết lập các giá trị khác nhau của các đầu vào: F0, F1, A (To A-bus), B (To B-bus) và Carry-in (Carry-in được nối với phần tử Constant có giá trị 0). Đồng thời quan sát giá trị trên các đầu ra: Output và Carry-Out (trong vi kiến trúc mà chúng ta đang từng bước xây dựng, chúng ta không quan tâm đến Carry-out của ALU-16bits, vì vậy trên hình 5.4 nó không được nối với bất cứ phần tử nào).




Hình 5.4 ALU 16 bit được xây dựng từ 2 ALU 8 bit

## 5.6. Bài tập

**Bài tập 5.1:** Hãy sử dụng các các phần tử "Hex Digit Display" và "Splitter" để đọc được dưới dạng số Hexa các giá trị đưa vào ALU 8 bit và kết quả nhận được ở đầu ra ALU 8 bit đã xây dựng khi làm bài tập 4 của "Bài tập mô phỏng số 3: Học sử dụng Logisim theo trợ giúp – Help".

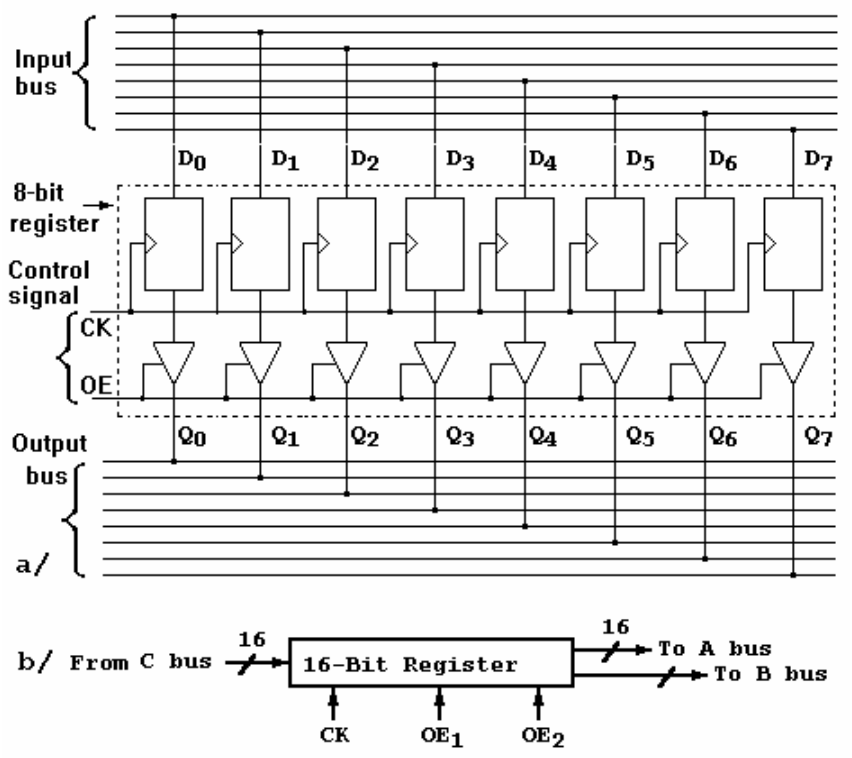
# Bài số 6: Xây dựng khối 16 thanh ghi của đường dữ liệu

## 6.1. Các yêu cầu và nội dung chính

- Xây dựng và mô phỏng sự hoạt động của khối các thanh ghi "16 Registers" của vi kiến trúc được mô tả trên hình 4-07 và hình 4-09 trong GT KTMT. Khối này gồm 16 thanh ghi, mỗi thanh ghi có kích thước16 bits, thứ tự các thanh ghi từ 0..15 là: PC, AC, SP, IR, TIR, 0, +1, -1, AMASK, SMASK, A, B, C, D, E, F.
- Mỗi một trong 16 thanh ghi nói trên có thể được điều khiển để nhận dữ liệu từ Bus-C bằng tín hiệu từ trường ENC trong thanh ghi MIR và việc nhận dữ liệu từ Bus-C chỉ có thể xảy ra trong chu kỳ con thứ 4 của tín hiệu đồng hồ. Việc chọn 1 trong 16 thanh ghi thực hiện bằng "C decoder", còn đầu vào 4 bit của "C decoder" nối với trường C của thanh ghi MIR. (Các thanh ghi 0, +1, -1, AMASK, SMASK chứa giá trị định trước nên sẽ không được nhận giá trị từ Bus-C).
- Mỗi một trong 16 thanh ghi nói trên có thể được điều khiển để đổ nội dung của nó ra Bus-A, hoặc Bus-B, hoặc đồng thời ra cả Bus-A và Bus-B.

## 6.2. Phân tích bài toán

Cách tạo khối các thanh ghi này được mô tả tại mục "4.1.1 Thanh ghi" và mục "4.1.2 Bus" của GT KTMT và được minh hoạ trên hình 6.1 dưới đây. Chú ý rằng hình 6.1a này mô tả cách tạo 1 thanh ghi kích thước 8 bits từ 8 thanh ghi 1 bit; hình 6.1b mô tả các bus và các tín hiệu nối với thanh ghi 16 bits.




Hình 6.1 a/ Thanh ghi 8 bit nối với một bus vào (input bus) và một bus ra (output bus). b/ Ký hiệu của một thanh ghi 16 bit với một bus vào và hai bus ra (Hình 4-02 trong GT KTMT)

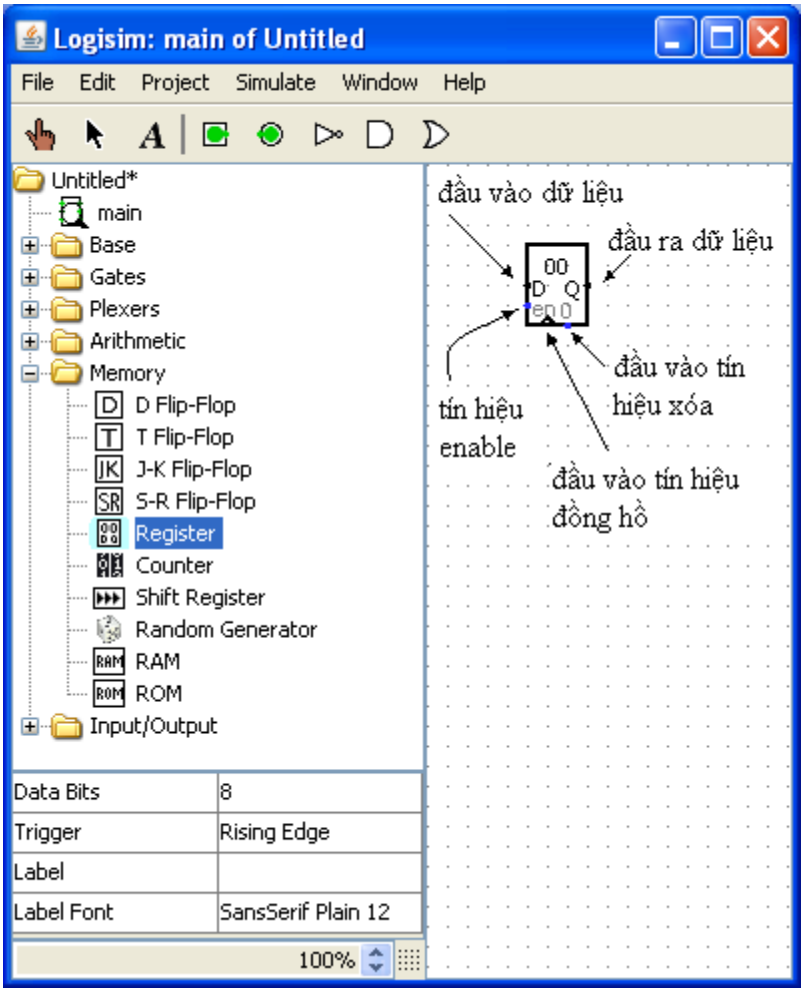
Trong bài thực hành này chúng ta sẽ sử dụng phần tử "register" thuộc nhóm "Memory" trong thư viện của Logisim. Một số thuộc tính của phần tử này có thể thay đổi được thông qua cửa sổ "Attribute Table" ở góc dưới bên trái cửa sổ chính của Logisim. Ý nghĩa của các thuộc tính như sau:

Data bits: Từ 1 đến 32, chúng ta chọn bằng 16.

Trigger: Tín hiệu đồng hồ kích hoạt phần tử nhớ (nhận input hoặc đưa nội dung ra output) (đầu vào có ghi đầu mũi tên), có thể chọn 1 trong 4 tín hiệu:

- Rising Edge: phần tử nhớ bị kích hoạt bằng sườn lên (dương) của xung kích
- Falling Edge: phần tử nhớ bị kích hoạt bằng sườn xuống (âm) của xung kích
- High Level: phần tử nhớ bị kích hoạt bằng mức cao của xung kích
- Low Level: phần tử nhớ bị kích hoạt bằng mức thấy của xung kích.

Chúng ta chọn "High Level", vì chúng ta muốn các thanh ghi được kích hoạt bởi xung đồng hồ có mức cao. (Các bạn sinh viên khi thực hành có thể thử các giá trị khác và quan sát tác động của các giá trị này)




Hình 6.2 Chức năng các "chân" của phần tử Register

Label: nhãn mà chúng ta muốn đặt cho phần tử nhớ. Chúng ta có thể đặt nhãn khác nhau cho các thanh ghi khác nhau, thí dụ: thanh ghi có đầu "en" nối với đầu ra 0 của decoder nên gán nhãn là PC, tương tự: 1: AC, 2: SP, ...

Label Font: font chữ và cỡ chữ của nhãn; Chúng ta cần chọn sao cho đẹp, chữ không đè lên các nét của hình vẽ.

Các đầu vào/ra của phần tử nhớ:

- D (Data bits): đầu vào dữ liệu (1..32 bits).
- Q (đầu ra dữ liệu): có độ rộng (số bit) bằng đầu vào D.
- "^": đầu vào (1 bit) của tín hiệu đồng hồ.
- en (Enable): tín hiệu mở chip chứa phần tử nhớ, tương tự tín hiệu CS (Chip Select) hoặc CE (Chip Enable) mà chúng ta đã gặp khi phân tích một số mạch điện trong "Chương 3 Mức logic số".
- "0": đầu vào tín hiệu xóa (đặt giá trị 0) nội dung phần tử nhớ.

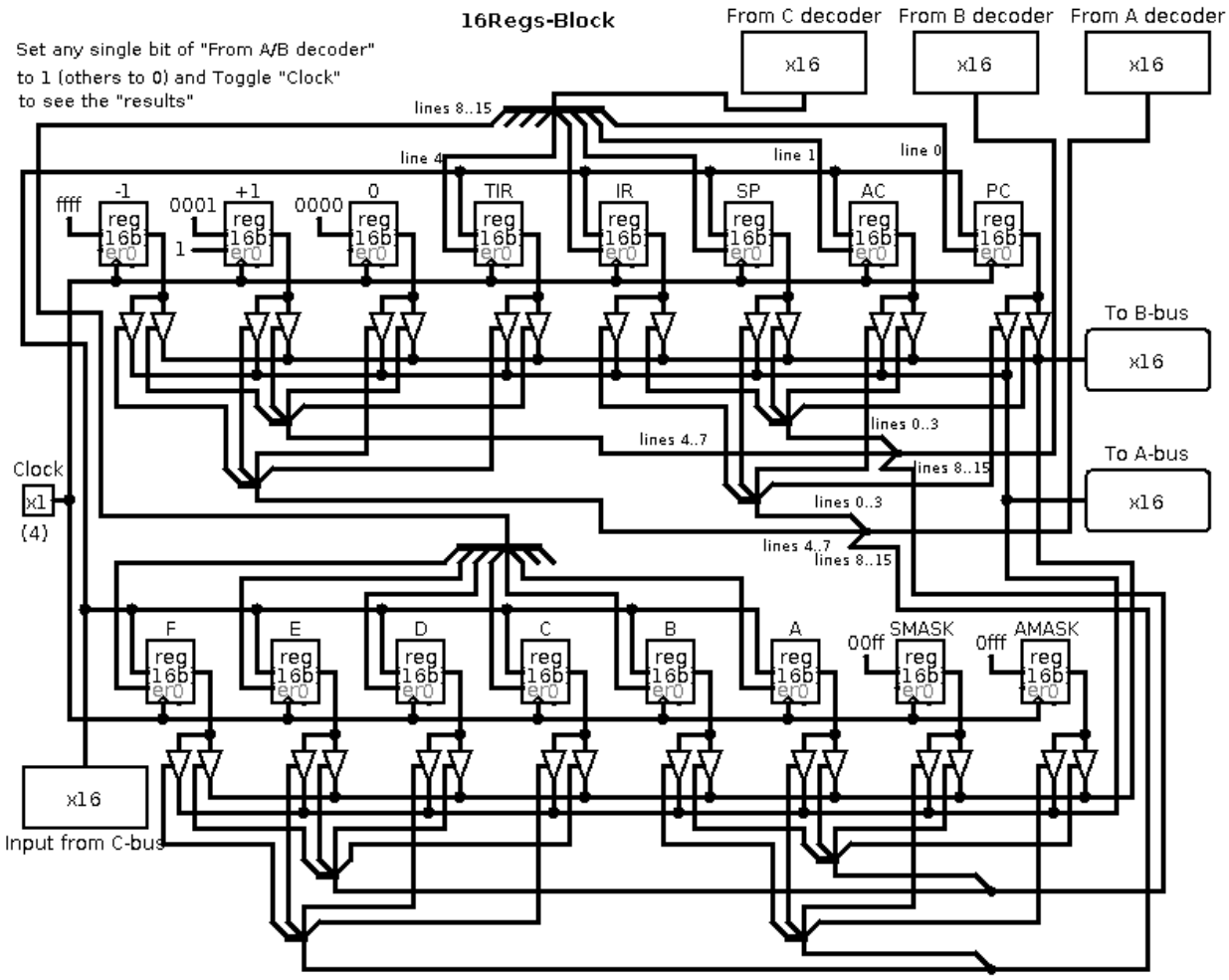
Sau khi biết được các thuộc tính và các đầu vào/ra của phần tử nhớ "Register", chúng ta sẽ thực hiện xây dựng khối 16 thanh ghi theo cách như sau (Việc phân tích cần đối chiếu với một số hình vẽ trong giáo trình Kiến trúc máy tính):

- Đầu vào D (16 dây) của 16 thanh ghi (trừ các thanh ghi "0", "+1", "-1", AMASK, SMASK) nối chung lại với nhau (các đường truyền bit 0 nối với nhau, các đường truyền bit 1 nối với nhau...) và sẽ nối với Bus-C thông qua phần tử "input" 16 bit (Xem Hình 4-02 và Hình 4-07 trong GT KTMT).
- Đầu ra Q (16 dây) của mỗi một trong số 16 phần tử "Register" được nối với 2 đầu vào của 2 bộ đệm 3 trạng thái (phần tử "Controlled buffer" thuộc nhóm "Gates"), để có thể được điều khiển để đổ dữ liệu ra hoặc Bus-A, hoặc Bus-B hoặc đồng thời ra cả 2 Bus-A và Bus-B. 16 đầu ra của 16 bộ đệm 3 trạng thái thứ nhất nối với nhau và nối với một phần tử "output" để đổ dữ liệu ra Bus-A; 16 đầu ra của 16 bộ đệm 3 trạng thái thứ hai nối với nhau và nối với một phần tử "output" để đổ dữ liệu ra Bus-B. Phần tử "Controlled buffer" có đầu ra nối với Bus-A được điều khiển bởi tín hiệu từ một trong các đầu ra của A-decoder. Tương tự như vậy, phần tử "Controlled buffer" có đầu ra nối với Bus-B được điều khiển bởi tín hiệu từ một trong các đầu ra của B-decoder (Xem Hình 4-02 và Hình 4-07).
- Đầu vào tín hiệu đồng hồ "^" của tất cả 16 thanh ghi được nối với nhau (Xem Hình 4-02).
- Mỗi đầu vào en (Enable) của một trong 16 phần tử "register" nối với một đầu ra của C-decoder. Như vậy C-decoder sẽ chọn lấy 1 trong 16 thanh ghi để nhận dữ liệu từ Bus-C vào.
- Đầu vào tín hiệu xóa "0": trong các bài thực hành này, chúng ta chưa sử dụng đến nên để hở (không nối đi đâu cả).

## 6.3. Xây dựng và mô phỏng khối 16-Register

### 6.3.1 Xây dựng khối 16-Register

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là 16registers-block. Trong cửa sổ con ở góc trên bên trái, ngoài tên main, ALU-1bit, ALU-8bits, ALU-16bits Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là 16registers-block.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con 16registers-block sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con "Canvas Window" của Logisim, chúng ta tạo một tập 16 thanh ghi như ở hình 6.3 dưới đây.
- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc (Exit) Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).




Hình 6.3 Khối 16 thanh ghi và các đầu vào, ra, điều khiển và đồng hồ

### 6.3.2 Mô phỏng hoạt động của khối 16-Register

Chúng ta có thể mô phỏng để xem đơn vị 16-Registers mà chúng ta vừa tạo ra có thực hiện đúng các chức năng như ý đồ thiết kế không.

Chọn 1 trong 16 thanh ghi để nhận dữ liệu vào từ Bus-C (trừ các thanh ghi "0", "+1", "-1", AMASK, SMASK có đầu vào D được nối cố định với các phần tử Constant có giá trị 0, +1, -1, 0FFF, 00FF):

1. Đặt giá trị cho phần tử input "clock" bằng 1 (sử dụng "Poke tool").
2. Đặt giá trị cho phần tử input "Input from C-bus" bằng một giá trị nhị phân nào đó, chẳng hạn bằng "1111.0000.1100.1100".
3. Đặt giá trị cho phần tử input "From C-decoder" bằng "0000.0000.0000.0001" để chọn thanh ghi "PC", cho nó nhận dữ liệu từ phần tử input "Input from C-bus". Nếu thay đổi giá trị này (chú ý: luôn chỉ có 1 bit bằng 1, còn các bit khác bằng 0) thì một trong 15 thanh ghi khác được chọn.

Chú ý: Với mạch điện và các giá trị được thiết lập như trên, chúng ta thường không đọc được giá trị nhị phân chứa vào thanh ghi được chọn. Để có thể nhìn thấy, chúng ta thực hiện thêm 2 bước như sau:

1. Đặt giá trị cho phần tử input "From B-decoder" giá trị nhị phân giống giá trị đặt cho phần tử input "From C-decoder".
2. Dùng phần tử "poke took" đảo giá trị của phần tử input "clock" 2 lần; chúng ta sẽ nhìn thấy giá trị mà thanh ghi được chọn đã nhận từ Bus-C, nay truyền ra phần tử output "To B-bus".

## 6.4. Bài tập

**Bài tập 6.1:** Sử dụng các các phần tử "Hex Digit Display" và "Splitter" để đọc được dưới dạng số Hexa các giá trị đưa vào các phần tử ouput "To A-Bus" và "To B-Bus".

# Bài số 7: Xây dựng các thanh ghi MAR và MBR của đường dữ liệu

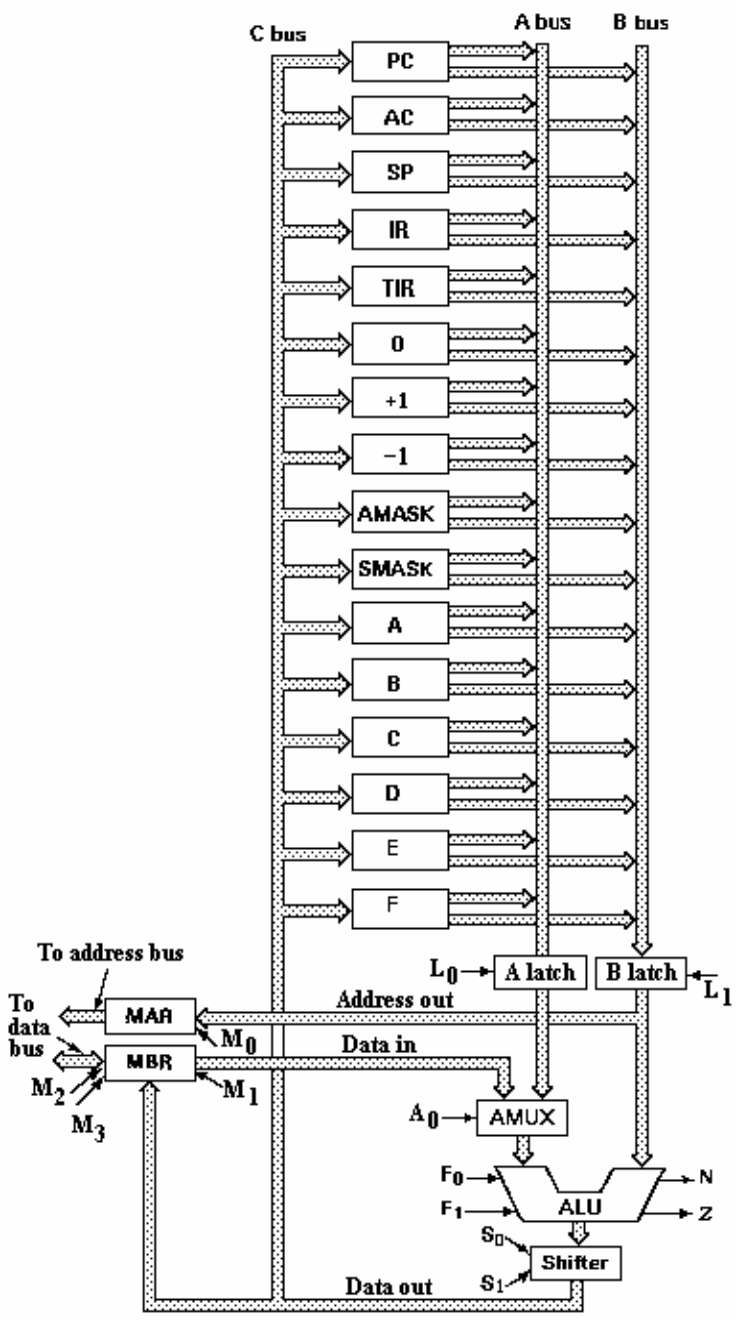
## 7.1. Các yêu cầu và nội dung chính

Đường dữ liệu (Data Path) được trình bày trong "Chương 4 Mức Vi chương trình" và được minh họa trên hình 7.1 (Hình 4-07 trong GT KTMT).

Trong 2 bài thực hành trước (bài 5 và bài 6), chúng ta đã xây dựng và mô phỏng sự làm việc của 2 thành phần của đường dữ liệu là ALU-16bits và khối 16 thanh ghi.

Trong bài thực hành này, chúng ta sẽ xây dựng và mô phỏng sự hoạt động của 2 thành phần khác nữa của đường dữ liệu, đó là 2 thanh ghi:

- MAR (Memory Address Register): Thanh ghi địa chỉ bộ nhớ.
- MBR (Memory Buffer Register): Thanh ghi đệm (dữ liệu) bộ nhớ.




Hình 7.1 Đường dữ liệu của một vi kiến trúc

## 7.2 Xây dựng và mô phỏng sự hoạt động của MAR

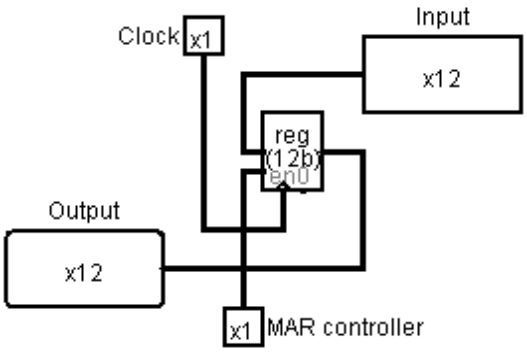
### 7.2.1 Phân tích

Chức năng của thanh ghi chốt địa chỉ bộ nhớ MAR được mô tả tại mục "4.1.6 Bộ nhớ chính" (GT KTMT). Trên hình vẽ minh họa đường dữ liệu ở đầu Bài tập mô phỏng số 7 này, tín hiệu điều khiển $M_0$ tích cực (=1) làm cho MAR chốt dữ liệu ở đầu vào nối với đầu ra B Latch của nó, khi $M_0$ đảo giá trị (từ 1 về 0), nội dung MAR không thay đổi theo tín hiệu vào và luôn được truyền ra đầu ra (chính là Address Bus, thuộc bus hệ thống của máy tính). Điều đáng lưu ý khi thực hành mô phỏng là, đầu ra và đầu vào của MAR đều truyền giá trị 12 bits chứ không phải 16 bits. 12 bits đưa vào đầu vào của MAR lấy từ 12 bits thấp trong số 16 bits truyền từ đầu ra của B Latch. Để tách 12 bits này chúng ta sẽ sử dụng phần tử splitter thuộc nhóm Wiring có trong thư viện của Logisim, khi chúng ta lắp ghép toàn bộ vi kiến trúc.

Ngoài tín hiệu điều khiển $M_0$ nói trên phải có mức tích cực, chúng ta đã biết rằng, MAR chỉ được chốt dữ liệu trong chu kỳ con thứ 3 của đơn vị đồng hồ (vấn đề này được phân tích trong mục "4.2.3 Việc định thời vi chỉ thị", GT KTMT). Tóm lại, có 2 tín hiệu điều khiển MAR chốt dữ liệu, một là tín hiệu lấy từ bit MAR trong thanh ghi MIR (chính là $M_0$ trong hình vẽ ở đầu bài thực hành này); hai là tín hiệu lấy từ đường chu kỳ con thứ 3 của đơn vị đồng hồ.

### 7.2.2 Xây dựng thanh ghi MAR

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là MAR-12bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là MAR-12bits.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con MAR-12bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con "Canvas Window" của Logisim, chúng ta xây dựng MAR-12bits như ở hình 7.2 dưới đây. Tất cả các phần tử cần sử dụng chúng ta đều đã biết.
- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).




Hình 7.2 Thanh ghi MAR và các đầu vào, ra, điều khiển và tín hiệu đồng hồ

### 7.2.3 Mô phỏng hoạt động của MAR

Chúng ta có thể mô phỏng để xem thanh ghi chốt MAR mà chúng ta vừa tạo ra có thực hiện đúng các chức năng như ý đồ thiết kế không. Cần chú ý là MAR có 2 tín hiệu điều khiển là Clock và $M_0$. Các bước thực hiện như sau:

1. Ban đầu, bit "Clock" và "$M_0$" bằng 0, chúng ta đặt giá trị cho phần tử "Input" bằng một giá trị nhị phân nào đó, chẳng hạn bằng "1111.0000.1100".
2. Thiết lập giá trị cho bit "Clock" và "$M_0$" đồng thời bằng 1, chúng ta sẽ thấy đầu ra "Output" sẽ có giá trị đúng bằng "Input" là: "1111.0000.1100". Hãy thử chỉ cho một trong hai tín hiệu "Clock" và "Mo" bằng 1, giá trị còn lại bằng 0, chúng ta sẽ thấy thanh ghi không thể nhận giá trị "Input".
3. Đảo bit "Clock" bằng 0.
4. Thay đổi giá trị của "Input", chúng ta sẽ thấy "Output" không bị thay đổi.

## 7.3 Xây dựng và mô phỏng sự hoạt động của MBR

### 7.3.1 Phân tích

Chức năng của thanh ghi đệm dữ liệu bộ nhớ MBR được mô tả tại mục "4.1.6 Bộ nhớ chính" và mục "4.2.1 Đường dữ liệu (Data path)" của GT KTMT.

Trên hình vẽ minh họa đường dữ liệu ở đầu Bài tập mô phỏng số 7 này, chúng ta thấy có 3 tín hiệu điều khiển MBR là $M_1$, $M_2$ và $M_3$:

- $M_1$: điều khiển việc nạp của MBR từ đầu ra của Shifter ($M_1$ =1: nạp). Trong "Hình 4-09 Sơ đồ đầy đủ của một vi kiến trúc" của GT KTMT, $M_1$ được nối với **bit MBR** của MIR.
- $M_2$: điều khiển việc đọc bộ nhớ, $M_2$ được nối với **bit RD** của MIR. Khi đọc, dữ liệu từ bộ nhớ được chứa vào MBR và truyền ra một trong hai đầu vào của AMUX. Dữ liệu từ đầu vào này có thể được điều khiển để đi ra đầu ra của AMUX và nạp vào đầu vào bên trái của ALU.
- $M_3$: điều khiển việc ghi bộ nhớ, $M_3$ được nối với **bit WR** của MIR. Khi ghi, dữ liệu đang chứa trong MBR được đưa ra bus dữ liệu nối với Main memory (bus bên trái MBR).

Ngoài 3 tín hiệu điều khiển nói trên, chúng ta đã biết rằng, MBR chỉ được chốt dữ liệu vào hoặc đưa dữ liệu ra trong chu kỳ con thứ 4 của đơn vị đồng hồ (vấn đề này được phân tích trong mục "4.2.3 Việc định thời vi chỉ thị", GT KTMT).

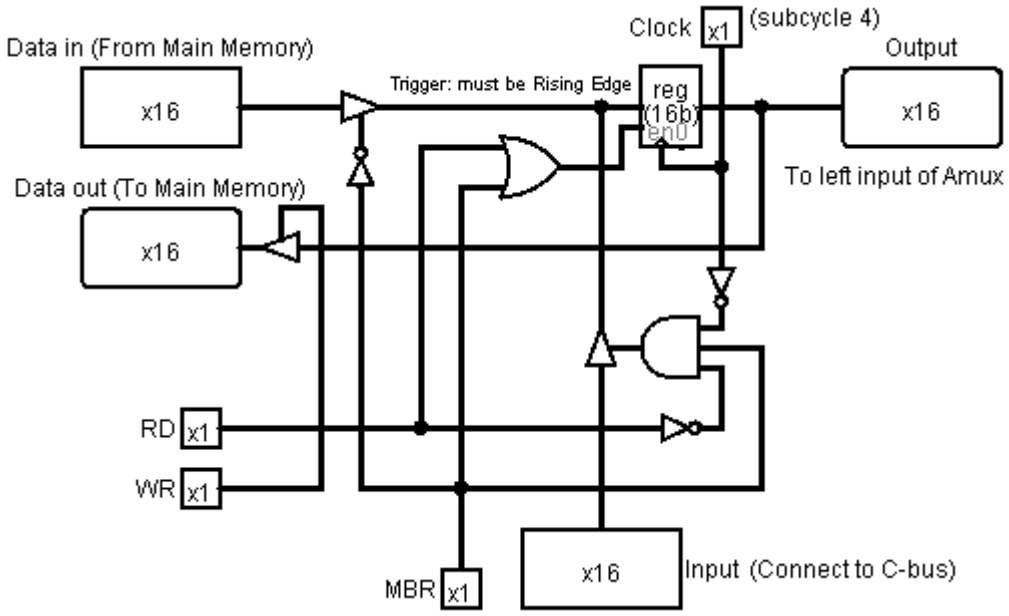
### 7.3.2 Xây dựng thanh ghi MBR

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là MBR-16bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là MBR-16bits.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con MBR-16bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.

- Trong cửa sổ con "Canvas Window" của Logisim, chúng ta xây dựng MBR-16bits như ở hình 7.3 dưới đây. Tất cả các phần tử cần sử dụng chúng ta đều đã biết.

  **Chú ý:**

  Bus dữ liệu nối MBR với Main memory là bus 2 chiều; tuy nhiên vì Logisim không có phần tử nào vừa có thể là input vừa có thể là output, cho nên chúng ta đành phải tạo ra 2 bus một chiều để thay thế.




Hình 7.3 Thanh ghi MBR và các đầu vào, ra, điều khiển và tín hiệu đồng hồ

- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).

### 7.3.3 Mô phỏng hoạt động của MBR

Chúng ta có thể mô phỏng để xem thanh ghi MBR mà chúng ta vừa tạo ra có thực hiện đúng các chức năng như ý đồ thiết kế không. Việc thực hiện có nhiều bước tương tự như đối với thanh ghi MAR.

Mô phỏng việc chứa dữ liệu từ Bus-C vào MBR theo các bước như sau:

1. Ban đầu, cần thiết lập giá trị cho:
   - MBR, RD, WR và Clock: 0 (có thể chọn từ menu: simulate \ Reset Simulation).
   - "Input (Connect to C-bus)": một giá trị nhị phân nào đó, chẳng hạn bằng "1111.0000.1100.1100".
   - Chọn "Simulation Enable" (chọn từ menu: Simulate \ Simulation Enable)
2. Thiết lập giá trị cho bit MBR bằng 1; sau đó đảo giá trị bit Clock từ 0 thành 1, chúng ta sẽ thấy đầu ra "Output" sẽ có giá trị đúng bằng "Input" là: "1111.0000.1100.1100".

Mô phỏng việc chứa dữ liệu từ bộ nhớ vào MBR (đọc bộ nhớ) theo các bước như sau:

1. Ban đầu, để tránh nhầm lẫn, nên reset lại mô phỏng (menu: simulate \ Reset Simulation).
2. Thiết lập giá trị cho phần tử input "Data in (From Main Memory)": một giá trị nhị phân nào đó, chẳng hạn bằng "1111.0000.1100.1100".
3. Thiết lập giá trị cho phần tử input RD bằng 1; sau đó đảo giá trị bit Clock từ 0 thành 1, chúng ta sẽ thấy đầu ra "Output" sẽ có giá trị đúng bằng "Input" là: "1111.0000.1100.1100".

Mô phỏng việc đưa dữ liệu từ MBR ra bộ nhớ (ghi bộ nhớ):

Sau việc mô phỏng trên (chứa dữ liệu từ bộ nhớ vào MBR), phần tử "Output" đang chứa giá trị "1111.0000.1100.1100", còn phần tử output "Data out (To Main Memory)" chưa từng được gán giá trị nên được Logisim coi là chứa giá trị không xác định và hiển thị nội dung phần tử này bằng 16 ký tự "x".

Chúng ta thực hiện mô phỏng việc ghi bộ nhớ theo như sau:

Thiết lập giá trị cho bit MBR bằng 0 và bit WR bằng 1, chúng ta sẽ thấy đầu ra "To Main Memory" sẽ có giá trị đúng bằng "Output" là: "1111.0000.1100.1100".

## 7.4. Bài tập

**Bài tập 7.1:** Với mạch con MBR-16bits như chúng ta đã xây dựng, việc ghi bộ nhớ chỉ đòi hỏi bit điều khiển WR bằng 1, chứ không cần "Clock (subcycle 4)" bằng 1. Hãy sửa lại mạch con đó sao cho dữ liệu từ MBR chỉ đưa ra được bus dữ liệu nối với Main Memory khi bit điều khiển "Clock (subcycle 4)" bằng 1.

# Bài số 8: Xây dựng các thành phần còn lại của đường dữ liệu

## 8.1. Các yêu cầu và nội dung chính

Đường dữ liệu (Data Path) được trình bày trong "Chương 4 Mức Vi chương trình, GT KTMT" và được minh họa trên hình 7.1 (Hình 4-07, GT KTMT).

Trong 3 bài thực hành trước (bài 5, 6 và 7), chúng ta đã xây dựng và mô phỏng sự làm việc của 4 thành phần là ALU-16bits, khối 16 thanh ghi, MAR và MBR.

Trong bài thực hành này, chúng ta sẽ xây dựng và mô phỏng sự hoạt động của các thành phần còn lại của đường dữ liệu, đó là:

- Shifter
- A Latch, B Latch
- AMUX

## 8.2. Xây dựng và mô phỏng sự hoạt động của shifter

### 8.2.1 Phân tích

- Thanh ghi dịch (Shifter) mà chúng ta cần xây dựng thực hiện một trong 3 thao tác (phép tính) theo tín hiệu điều khiển 2 bit được ký hiệu là S1S0.

| S1S0 | Thao tác |
|---|---|
| 00 | No Shift (không dịch bit) |
| 01 | Shift Left (dịch trái 1 vị trí bit) |
| 10 | Shift Right (dịch phải 1 vị trí bit) |
| 11 | Không sử dụng giá trị này |

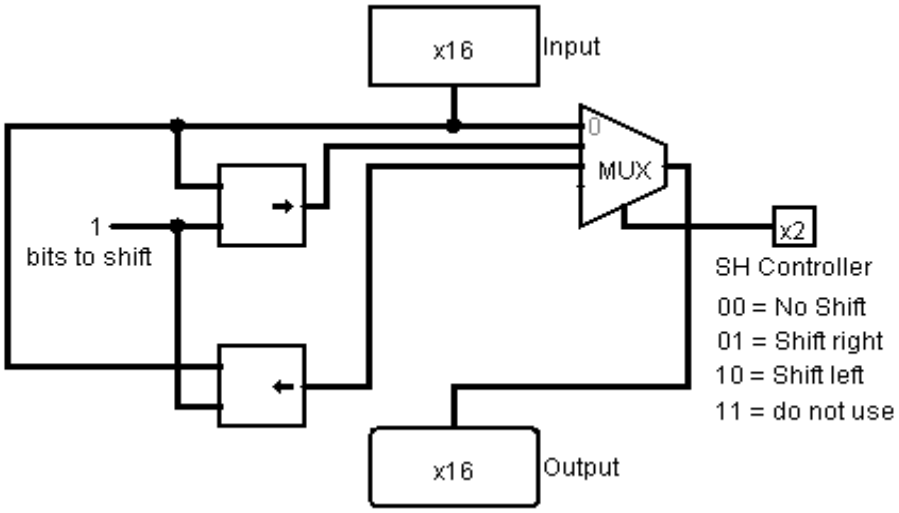
  Khi dịch trái/phải, giá trị 0 được đưa vào vị trí bit 0/15. Bit 15/0 bị "bật ra" khi dịch trái/phải nhưng (trong môn học này) chúng ta không cần quan tâm.

- Đầu vào (16 bits) của shifter nối với đầu ra của ALU-16bits.
- Đầu ra (16 bits) của shifter nối với đầu vào của MBR và C-Bus. Điều này có nghĩa là kết quả có thể đưa ra bộ nhớ chính (Main memory) thông qua MBR, hoặc đưa vào 1 trong 16 thanh ghi qua C-Bus để nó tham gia vào phép tính khác.

### 8.2.2 Xây dựng mạch con shifter

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Shifter-16bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main, ALU-1bit, ALU-8bits, ALU-16bits, 16registers-block v.v. Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Shifter-16bits.

- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Shifter-16bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con "Canvas Window" của Logisim, chúng ta xây dựng Shifter-16bits như ở hình 8.1 dưới đây.




Hình 8.1 Đơn vị Shifter gồm 2 bộ left, right Shifter và đầu vào, ra, điều khiển

- Ngoài các phần tử input và output đã biết, chúng ta sử dụng phần tử shifter trong nhóm Arithmetic và phần tử MUX (Multiplexer) trong nhóm Plexers của thư viện trong Logisim.
- Phần tử shifter:
  - Số thuộc tính: 2 ("Data Bits" và "Shift Type"), chúng ta cần thiết lập giá trị của chúng cho phù hợp với yêu cầu của mình:
    + Data Bits: 16 (có thể nhận các giá trị trong miền 1..32)
    + Shift Type: Logical Left (các giá trị khác có thể nhận là: Logical Right, Arithmetic Right, Roll Left, Roll Right)
  - Số chân: 3 (2 chân cho tín hiệu vào, 1 chân cho tín hiệu ra):
    + 1 chân vào cho data (16 bits): phía trên bên trái.
    + 1 chân ra cho data (16 bits): phía bên phải, có đánh dấu mũi tên, hướng mũi tên tùy theo chúng ta chọn thuộc tính "Shift Type" là left hay right.
    + 1 chân vào để điều khiển số vị trí bit sẽ dịch, trên hình 8.1 có ghi nhãn "bits to shift". Nếu chúng ta đã thiết lập "Data Bits" =16 (2^4) thì đầu vào này nhất thiết phải là 4 bit. Nếu chúng ta chỉ muốn dịch 1 bit, thì thiết lập giá trị cho phần tử input nối với đầu vào này bằng 0001.
- Phần tử MUX (Multiplexer):
  - Số thuộc tính: 6, chúng ta cần thiết lập giá trị của chúng cho phù hợp với yêu cầu của mình:
    + Facing: East - hướng đầu ra về phía đông (các giá trị khác: West, North, South)

+ Select Location: Chọn vị trí của đầu vào điều khiển "Select" (các giá trị: Bottom/Left, Top/Right)
+ Select Bits: 2 (vì chúng ta cần chọn 1 trong 3 kết quả để đưa ra)
+ Data Bits: 16
+ Disable Output: Nếu "Include Enable"=Yes, thì lựa chọn "Disable Output" này dùng để chọn giá trị đầu ra của Multiplexer là Floating hoặc Zero.
+ Include Enable: Chọn có bổ sung chân điều khiển làm việc hay không làm việc (Yes/No) cho Shifter.

– Số chân: 6 (5 chân cho tín hiệu vào, 1 chân cho tín hiệu ra):
+ Chân vào số 0 (có ký hiệu 0): 16 bits cho giá trị "no shift".
+ Chân vào số 1 (bên dưới chân 0): 16 bits cho giá trị "shift right".
+ Chân vào số 2: 16 bits cho giá trị "shift left".
+ Chân vào số 3: không sử dụng đến.
+ Chân vào cho 2 bits điều khiển: ở trên cạnh bên của hình thang cân (ký hiệu của Multiplexer).
+ Chân ra: 16 bits, nằm ở cạnh đáy nhỏ của hình thang cân, có giá trị bằng 1 trong 3 đầu vào, được chọn bằng 2 bits điều khiển.

- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).

### 8.2.3 Mô phỏng hoạt động của shifter

Chúng ta có thể mô phỏng để xem đơn vị shifter mà chúng ta vừa tạo ra có thực hiện đúng các chức năng như ý đồ thiết kế không. Có thể thực hiện như sau:

1. Đặt giá trị cho phần tử "Input" bằng một giá trị nhị phân nào đó, chẳng hạn bằng "1111.0000.1100.1100".
2. Sử dụng phần tử "Poke tool" để thiết lập giá trị cho 2 bits điều khiển (SH Controller) lần lượt bằng 00, 01 và 10. Nếu chúng ta xây dựng đúng, tại đầu ra "Output" chúng ta sẽ thấy các giá trị đưa ra lần lượt là: "1111.0000.1100.1100", "1110.0001.1001.1000" và "0111.1000.0110.0110". Đó chính là kết quả "no shift", "shift left" và "shift right" đối với dữ liệu đưa vào.

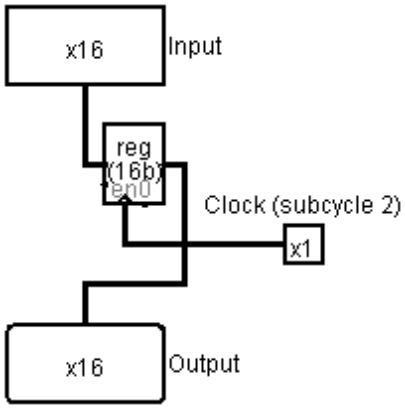
## 8.3. Xây dựng và mô phỏng sự hoạt động của các thanh ghi chốt A Latch và B Latch

### 8.3.1 Phân tích

Thanh ghi chốt mà chúng ta sẽ xây dựng để làm A Latch và B Latch có chức năng rất đơn giản. Nó có đầu vào và đầu ra 16 bits, đầu vào điều khiển 1 bit (L0 hoặc L1). Khi tín hiệu trên đầu vào điều khiển bằng 1, giá trị 16 bits ở đầu vào sẽ được nhớ và đưa ra đầu ra. Khi tín hiệu điều khiển từ 1 trở về 0, giá trị trong thanh ghi (và đầu ra) không thay đổi được nữa, dù giá trị ở đầu vào có thay đổi.

### 8.3.2 Xây dựng thanh ghi chốt (Latch)

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Latch-16bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main, ALU-1bit, ALU-8bits, ALU-16bits, 16registers-block, Shifter-16bits v.v. Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Latch-16bits.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Latch-16bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con "Canvas Window" của Logisim, chúng ta xây dựng Latch-16bits như ở hình vẽ 8.2 dưới đây. Tất cả các phần tử cần sử dụng chúng ta đều đã biết.




Hình 8.2 Đơn vị Latch Register gồm thanh ghi 16 bit, đầu vào, ra và tín hiệu đồng hồ

### 8.3.3 Mô phỏng hoạt động của thanh ghi chốt (Latch Register)

Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).

Chúng ta có thể mô phỏng để xem đơn vị latch mà chúng ta vừa tạo ra có thực hiện đúng các chức năng như ý đồ thiết kế không. Có thể thực hiện như sau:

1. Ban đầu, bit "Clock" bằng 0 và chúng ta đặt giá trị cho phần tử "Input" bằng một giá trị nhị phân nào đó, chẳng hạn bằng "1111.0000.1100.1100".

2. Thiết lập giá trị cho bit “Clock” bằng 1, chúng ta sẽ thấy đầu ra “Output” sẽ có giá trị đúng bằng “Input” là: “1111.0000.1100.1100”.
3. Đảo bit “Clock” bằng 0.
4. Thay đổi giá trị của “Input”, chúng ta sẽ thấy “Output” không bị thay đổi.

## 8.4. Xây dựng và mô phỏng sự hoạt động của AMUX

AMUX là một bộ dồn kênh 2-to-1, chúng ta sẽ sử dụng phần tử Multiplexer có sẵn trong nhóm Plexers trong thư viện của Logisim làm AMUX. Ngoài phần tử này, chúng ta không cần thêm một phần tử chức năng nào khác nữa.

## 8.5 Bài tập

**Bài tập 8.1:** Tạo và mô phỏng hoạt động của bộ dịch vòng (rotary shifter) bằng cách bổ sung các phần tử cần thiết vào mạch điện của bộ dịch có sơ đồ biểu diễn trên Hình 3-1 (GT KTMT). Phép toán dịch vòng trái (Rotate Left - ROL tương tự shift left, nhưng có điểm khác là bit bậc cao nhất vòng lại vị trí bit bậc thấp nhất. Rotate Right - ROR tương tự shift right, nhưng có điểm khác là bit bậc thấp nhất vòng lên vị trí bit bậc cao nhất.)

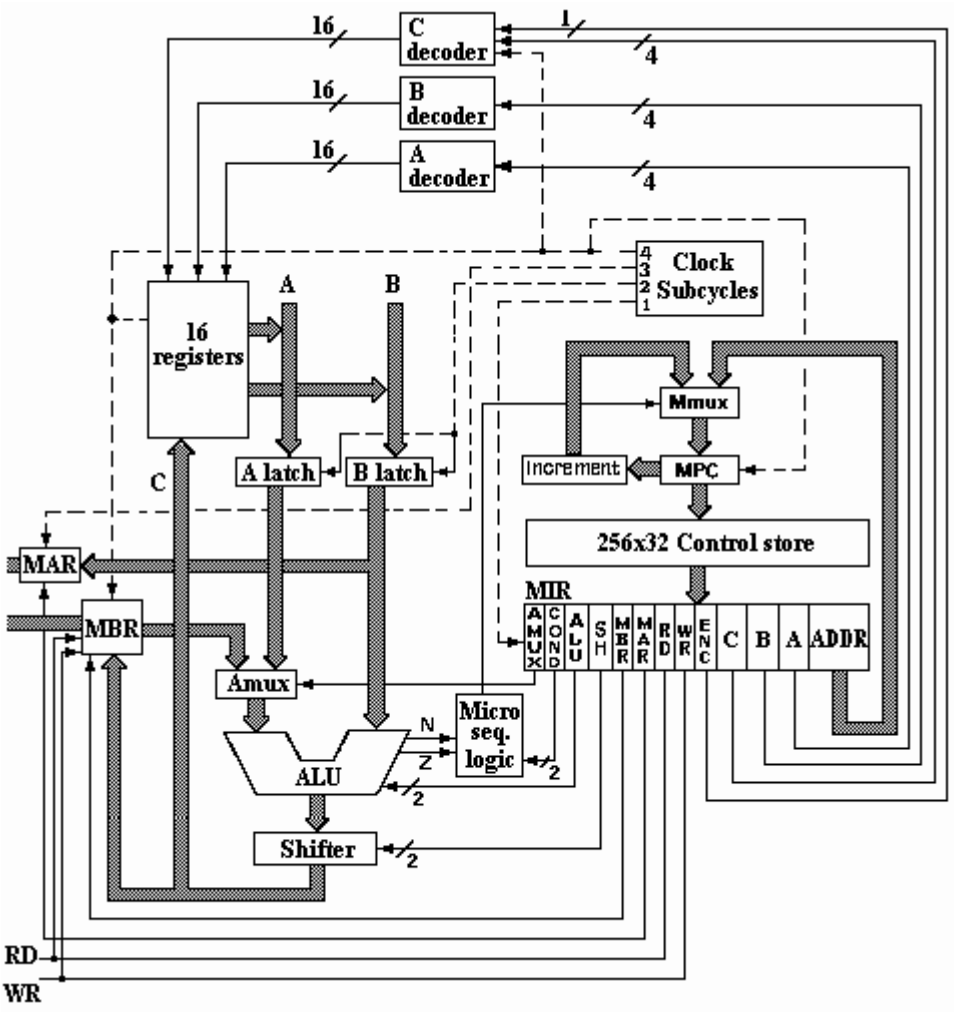
Chú ý: Bài tập 1 chương 3 (BT#08) đã yêu cầu tạo và mô phỏng bộ dịch có sơ đồ biểu diễn trên Hình 3-1 (GT KTMT).

# Bài số 9: Xây dựng Control Store của đơn vị điều khiển

## 9.1. Các yêu cầu và nội dung chính

Đơn vị điều khiển – CU (Control Unit) được giới thiệu khái quát trong mục "2.1 Kiến trúc chung của máy tính điện tử"; Tại chương 2 (GT KTMT), chúng ta đã biết rằng CU thực hiện việc điều khiển sự hoạt động của tất cả các đơn vị tạo nên hệ thống máy tính theo chương trình trong bộ nhớ chính.

CU được nghiên cứu sâu hơn trong chương 4, tại các mục "4.2.2 Vi chỉ thị - microinstruction", "4.2.3 Việc định thời vi chỉ thị", "4.2.4 Sự định trình tự các vi chỉ thị"... Trong sơ đồ khối đầy đủ của một vi kiến trúc dưới đây, chúng ta thấy CU bao gồm 10 đơn vị sau: (1) Control Store, (2) MIR, (3) Micro Seq, (4) MPC, (5) Increment, (6) Mmux, (7) A Decoder, (8) B decoder, (9) C decoder và (10) đơn vị tạo tín hiệu đồng hồ - Clock.




Hình 9.1 Sơ đồ khối đầy đủ của một vi kiến trúc (Hình 4-09, GT KTMT)

Trong Bài tập mô phỏng số 9 này chúng ta sẽ xây dựng và mô phỏng sự hoạt động của 1 trong số 10 đơn vị cấu thành CU, đó là Control Store (bộ nhớ điều khiển). Control Store phải chứa được toàn bộ vi chương trình thực hiện thông dịch các lệnh lấy về từ bộ nhớ chính (Main memory).

- Đầu vào: Có 1 đầu vào địa chỉ 8 bits nối với đầu ra của MPC. Địa chỉ này dùng để chọn 1 trong số 2^8 vi chỉ thị chứa trong Control Store.
- Đầu ra: Có 1 đầu ra 32 bits nối với đầu vào của MIR.

## 9.2. Xây dựng và mô phỏng sự hoạt động của Control Store

### 9.2.1 Phân tích

- Về nguyên tắc, Control Store có thể là ROM hoặc RAM. Trong bài thực hành này, chúng ta sẽ **sử dụng phần tử ROM** có trong thư viện của Logisim làm Control Store. Lý do: Trong bài thực hành số 10, chúng ta sẽ tạo ra và mô phỏng hoạt động của bộ nhớ chính (Main memory), chỉ có thể sử dụng phần tử RAM. Như vậy chúng ta sử dụng cả 2 loại phần tử nhớ trong thực hành.
- Kích thước: Trong mục "4.4 Thí dụ về một vi chương trình", chúng ta đã phân tích kỹ lưỡng một vi chương trình gồm 79 vi chỉ thị, mỗi vi chỉ thị dài 32 bit. Như vậy kích thước của Control Store tối thiểu là 79x32 bit. Tuy nhiên, để có thể mở rộng được, yêu cầu sinh viên thiết kế Control Store có kích thước **256x32 bits**.
- Từ kích thước 256x32 dẫn đến việc một số trong các thuộc tính của phần tử ROM làm Control Store bắt buộc phải có giá trị như sau:
  - Address Bit Width: 8 (để có thể chọn 1 trong 256)
  - Data Bit Width: 32 (kích thước 1 vi chỉ thị)
- Nội dung (Contents): Chính là vi chương trình được trình bày tại mục "4.4.2 Thí dụ về một vi chương trình", sinh viên cần nạp vào phần tử ROM, sau khi đã chuyển từ dạng MAL sang dạng nhị phân.

### 9.2.2. Xây dựng và chuẩn bị nội dung để nạp cho Control Store

Về việc xây dựng đơn vị chức năng Control Store, chúng ta có thể thực hiện theo một trong hai cách sau đây.

Cách thứ nhất: Chúng ta không xây dựng Control Store như một mạch con (subcircuit) như các mạch con mà chúng ta đã xây dựng ở nhiều bài thực hành trước mà sử dụng phần tử ROM thuộc nhóm phần tử Memory để xây dựng vi kiến trúc. Cách này có ưu/nhược điểm như sau:

- Ưu điểm: Khi mô phỏng vi kiến trúc chúng ta có thể nhìn thấy nội dung các ô nhớ trong Control Store đang được truy cập, vì vậy việc học (qua mô phỏng) là thuận lợi hơn.
- Nhược điểm: Diện tích mà phần tử này chiếm trên cửa sổ làm việc (Canvas Window) lớn hơn, có thể hạn chế việc chúng ta mở rộng vi kiến trúc.

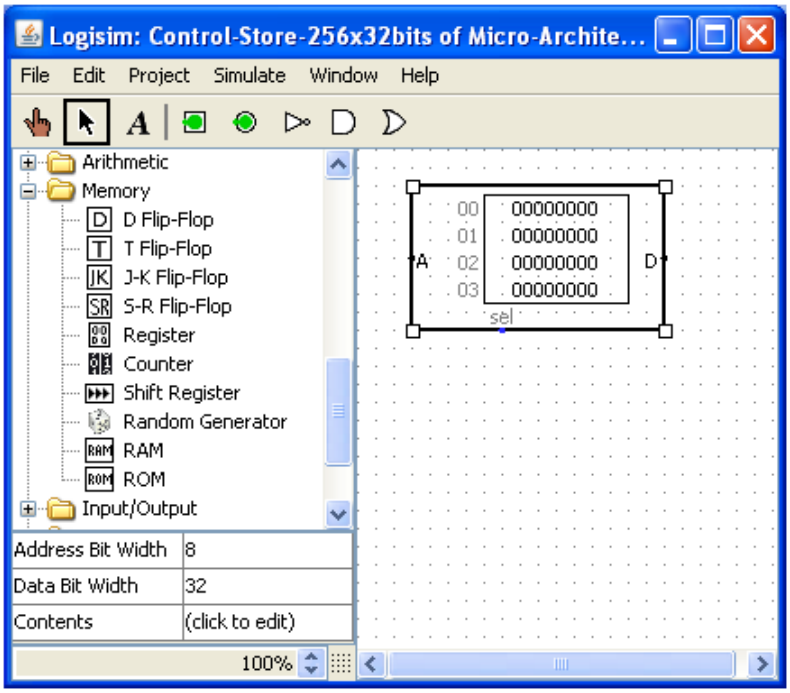
Do ưu điểm nêu trên là rất lớn cho người mới học thực hành mô phỏng bằng Logisim, nên chúng ta sẽ theo cách thứ nhất này khi xây dựng vi kiến trúc đầy đủ.

Cách thứ hai: Chúng ta xây dựng Control Store như một mạch con (subcircuit), theo cách mà chúng ta đã xây dựng các đơn vị chức năng ở nhiều bài thực hành trước. Cách này có ưu/nhược điểm như sau:

- Ưu điểm: Diện tích mà mạch con này chiếm trên cửa sổ làm việc (Canvas) nhỏ hơn, có thể tạo thuận lợi cho chúng ta khi cần mở rộng vi kiến trúc.
- Nhược điểm: Khi mô phỏng vi kiến trúc chúng ta không thể nhìn thấy nội dung các ô nhớ bên trong Control Store, vì vậy việc học (qua mô phỏng) không được thuận lợi.

Nếu theo cách thứ hai, chúng ta cần thực hiện lần lượt các bước sau:

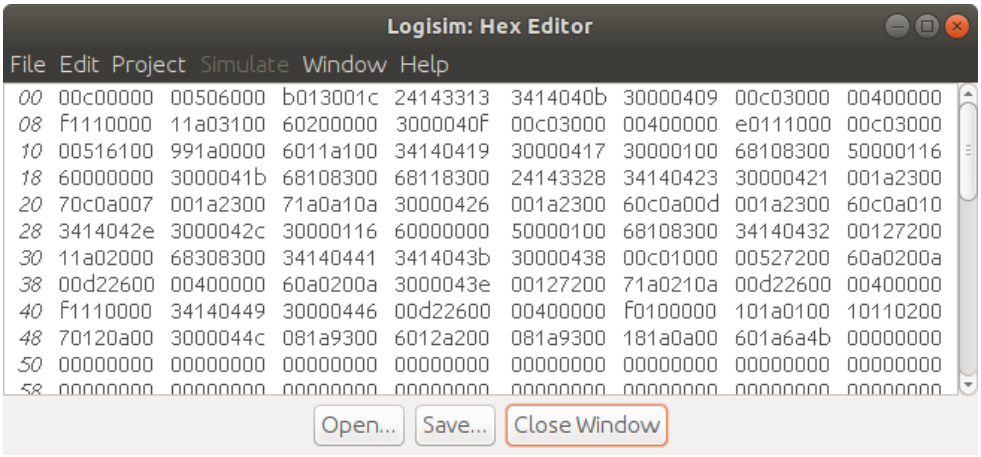
- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Control-Store-256x32bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Control-Store-256x32bits.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Control-Store-256x32bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con "Canvas Window" của Logisim, việc xây dựng Control-Store-256x32bits hết sức đơn giản:
  - Lấy một phần tử ROM trong nhóm Memory đặt vào "Canvas Window".
  - Thiết lập giá trị cho các thuộc tính của nó, như đã phân tích ở trên (Address Bit Width: 8; Data Bit Width: 32).




Hình 9.2 Phần tử ROM được lấy làm Control Store

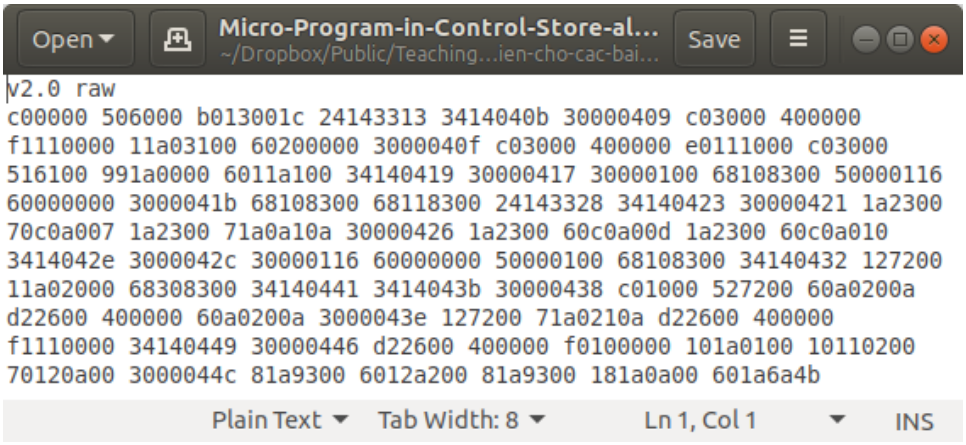
Nạp nội dung (vi chương trình) cho Control-Store:

- Để nạp nội dung cho phần tử nhớ Control-Store, hãy kích nút chuột bên phải, Logisim hiện lên một danh sách các lựa chọn. Hãy chọn "Edit Contents...", Logisim sẽ hiện lên một cửa sổ "Logisim: Hex Editor" để người sử dụng nạp nội dung cho phần tử nhớ Control-Store.




Hình 9.3 Cửa sổ "Hex Editor" để người sử dụng "nạp" nội dung từng ô nhớ Control Store

- Việc nạp nội dung cho Control-Store tuy đơn giản, nhưng mất nhiều thời gian và đòi hỏi tuyệt đối chính xác. Hình vẽ trên là nội dung Control-Store đã được nạp 79 (4FH = 4*16 + 15) words dưới dạng số Hexa, mỗi word 32 bits là một vi chỉ thị của vi chương trình làm nhiệm vụ thông dịch.




Hình 9.4 Nội dung Control-Store được export ra file và hiển thị bởi GEDIT

- Chúng ta có thể dùng một text editor, loại dùng để soạn thảo và/hoặc hiển thị file dạng "Plain text" (file chỉ chứa các ký tự ASCII chuẩn, không chứa ký tự điều khiển), chẳng hạn GEDIT (trong Ubuntu) để soạn nội dung cho Control-Store, sau đó sẽ nhập (import) vào Control-Store (chọn "Load Image" sau khi kích nút chuột phải vào phần tử ROM dùng làm Control-Store). Chúng ta cũng có thể Export (chọn "Save Image" sau khi kích nút chuột phải vào phần tử ROM dùng làm Control-Store) nội dung của nó ra file. Hình 9.4 là cửa sổ của GEDIT (một text editor trong Ubuntu) đang hiện nội dung file dạng "Plain text" chứa nội dung của Control-Store được export ra. Chú ý: Mỗi ô nhớ của Control-Store có kích thước 32 bits, ứng với số Hexa có 6 chữ số, nhưng khi Export ra file, các số 0 trước số khác không đầu tiên được bỏ qua.

## Vi chương trình được nạp vào Control Store

| Địa chỉ | Statement (Chỉ thị ngôn ngữ MAL) | AMUX | COND | ALU | SH | MBR | MAR | RD | WR | ENC | C | B | A | ADDR | Chỉ thị (Hexa) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 (00h) | mar := pc; rd | 0 | 00 | 00 | 00 | 0 | 1 | 1 | 0 | 0 | 0000 | 0000 | 0000 | 0000.0000 | 00.C0.00.00 |
| 1 (01h) | pc := pc+1; rd; | 0 | 00 | 00 | 00 | 0 | 0 | 1 | 0 | 1 | 0000 | 0110 | 0000 | 0000.0000 | 00.50.60.00 |
| 2 (02h) | ir := mbr; if n then goto 28; | 1 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 1 | 0011 | 0000 | 0000 | 0001.1100 | B0.13.00.1C |
| 3 (03h) | tir :=lshift(ir+ir); if n then goto 19; | 0 | 01 | 00 | 10 | 0 | 0 | 0 | 0 | 1 | 0100 | 0011 | 0011 | 0001.0011 | 24.14.33.13 |
| 4 (04h) | tir := lshift(tir); if n then goto 11; | 0 | 01 | 10 | 10 | 0 | 0 | 0 | 0 | 1 | 0100 | 0000 | 0100 | 0000.1011 | 34.14.04.0B |
| 5 (05h) | alu := tir; if n then goto 9; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0100 | 0000.1001 | 30.00.04.09 |
| 6 (06h) | mar := ir; rd; | 0 | 00 | 00 | 00 | 0 | 1 | 1 | 0 | 0 | 0000 | 0011 | 0000 | 0000.0000 | 00.C0.30.00 |
| 7 (07h) | rd; | 0 | 00 | 00 | 00 | 0 | 0 | 1 | 0 | 0 | 0000 | 0000 | 0000 | 0000.0000 | 00.40.00.00 |
| 8 (08h) | ac := mbr; goto 0 | 1 | 11 | 10 | 00 | 0 | 0 | 0 | 0 | 1 | 0001 | 0000 | 0000 | 0000.0000 | F0.11.00.00 |
| 9 (09h) | mar := ir; mbr := ac; wr; | 0 | 00 | 10 | 00 | 1 | 1 | 0 | 1 | 0 | 0000 | 0011 | 0001 | 0000.0000 | 11.A0.31.00 |
| 10 (0Ah) | wr; goto 0; | 0 | 11 | 00 | 00 | 0 | 0 | 0 | 1 | 0 | 0000 | 0000 | 0000 | 0000.0000 | 60.20.00.00 |
| 11 (0Bh) | alu := tir; if n then goto 15; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0100 | 0000.1111 | 30.00.04.0F |
| 12 (0Ch) | mar := ir; rd; | 0 | 00 | 00 | 00 | 0 | 1 | 1 | 0 | 0 | 0000 | 0011 | 0000 | 0000.0000 | 00.C0.30.00 |
| 13 (0Dh) | rd; | 0 | 00 | 00 | 00 | 0 | 0 | 1 | 0 | 0 | 0000 | 0000 | 0000 | 0000.0000 | 00.40.00.00 |
| 14 (0Eh) | ac := mbr + ac; goto 0; | 1 | 11 | 00 | 00 | 0 | 0 | 0 | 0 | 1 | 0001 | 0001 | 0000 | 0000.0000 | E0.11.10.00 |
| 15 (0FH) | mar := ir; rd; | 0 | 00 | 00 | 00 | 0 | 1 | 1 | 0 | 0 | 0000 | 0011 | 0000 | 0000.0000 | 00.C0.30.00 |
| 16 (10H) | ac := ac + 1; rd; | 0 | 00 | 00 | 00 | 0 | 0 | 1 | 0 | 1 | 0001 | 0001 | 0110 | 0000.0000 | 00.51.61.00 |
| 17 (11H) | a := inv(mbr); | 1 | 00 | 11 | 00 | 0 | 0 | 0 | 0 | 1 | 1010 | 0000 | 0000 | 0000.0000 | 98.1A.00.00 |
| 18 (12H) | ac := ac + a; goto 0; | 0 | 11 | 00 | 00 | 0 | 0 | 0 | 0 | 1 | 0001 | 1010 | 0001 | 0000.0000 | 60.11.A1.00 |
| 19 (13h) | tir := lshift(tir); if n then goto 25; | 0 | 01 | 10 | 10 | 0 | 0 | 0 | 0 | 1 | 0100 | 0000 | 0100 | 0001.1001 | 34.14.04.19 |
| 20 (14H) | alu := tir; if n then goto 23; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0100 | 0001.0111 | 30.00.04.17 |
| 21 (15H) | alu := ac; if n then goto 0; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0001 | 0000.0000 | 30.00.01.00 |
| 22 (16H) | pc := band(ir, amask); goto 0; | 0 | 11 | 01 | 00 | 0 | 0 | 0 | 0 | 1 | 0000 | 1000 | 0011 | 0000.0000 | 68.10.83.00 |
| 23 (17H) | alu := ac; if z then goto 22; | 0 | 10 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0001 | 0001.0110 | 50.00.01.16 |
| 24 (18H) | goto 0; | 0 | 11 | 00 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0000 | 0000.0000 | 60.00.00.00 |
| 25 (19h) | alu := tir; if n then goto 27; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0100 | 0001.1011 | 30.00.04.1B |
| 26 (1Ah) | pc := band(ir, amask); goto 0; | 0 | 11 | 01 | 00 | 0 | 0 | 0 | 0 | 1 | 0000 | 1000 | 0011 | 0000.0000 | 68.10.83.00 |
| 27 (1BH) | ac := band(ir, amask); goto 0; | 0 | 11 | 01 | 00 | 0 | 0 | 0 | 0 | 1 | 0001 | 1000 | 0011 | 0000.0000 | 68.11.83.00 |
| 28 (1CH) | tir := lshift(ir+ir); if n then goto 40; | 0 | 01 | 00 | 10 | 0 | 0 | 0 | 0 | 1 | 0100 | 0011 | 0011 | 0010.1000 | 24.14.33.28 |
| 29 (1DH) | tir := lshift(tir); if n then goto 35; | 0 | 01 | 10 | 10 | 0 | 0 | 0 | 0 | 1 | 0100 | 0000 | 0100 | 0010.0011 | 34.14.04.23 |
| 30 (1EH) | alu := tir; if n then goto 33; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0100 | 0010.0001 | 30.00.04.21 |
| 31 (1FH) | a := ir + sp; | 0 | 00 | 00 | 00 | 0 | 0 | 0 | 0 | 1 | 1010 | 0011 | 0010 | 0000.0000 | 00.1A.23.00 |
| 32 (20H) | mar := a; rd; goto 7; | 0 | 11 | 10 | 00 | 0 | 1 | 1 | 0 | 0 | 0000 | 1010 | 0000 | 0000.0111 | 70.c0.a0.07 |
| 33(21H) | a := ir + sp; | 0 | 00 | 00 | 00 | 0 | 0 | 0 | 0 | 1 | 1010 | 0010 | 0011 | 0000.0000 | 00.1a.23.00 |
| 34(22H) | mar := a; mbr := ac; wr; goto 10; | 0 | 11 | 10 | 00 | 1 | 1 | 0 | 1 | 0 | 0000 | 1010 | 0001 | 0000.1010 | 71.a0.a1.0a |
| 35(23H) | alu := tir; if n then goto 38; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0100 | 0010.0110 | 30.00.04.26. |
| 36(24H) | a := ir + sp; | 0 | 00 | 00 | 00 | 0 | 0 | 0 | 0 | 1 | 1010 | 0010 | 0011 | 0000.0000 | 00.1a.23.00 |
| 37(25H) | mar := a; rd; goto 13; | 0 | 11 | 00 | 00 | 0 | 1 | 1 | 0 | 0 | 0000 | 1010 | 0000 | 0000.1101 | 60.c0.a0.0d |
| 38(26H) | a := ir + sp; | 0 | 00 | 00 | 00 | 0 | 0 | 0 | 0 | 1 | 1010 | 0010 | 0011 | 0000.0000 | 00.1a.23.00 |
| 39(27H) | mar := a; rd; goto 16; | 0 | 11 | 00 | 00 | 0 | 1 | 1 | 0 | 0 | 0000 | 1010 | 0000 | 0001.0000 | 60.c0.a0.10 |

| Vi chương trình được nạp vào Control Store | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Địa chỉ | Statement (Chỉ thị ngôn ngữ MAL) | AMUX | COND | ALU | SH | MBR | MAR | RD | WR | ENC | C | B | A | ADDR | Chỉ thị (Hexa) |
| 40(28H) | tir := lshift(tir); if n then goto 46; | 0 | 01 | 10 | 10 | 0 | 0 | 0 | 0 | 1 | 0100 | 0000 | 0100 | 0010.1110 | 34.14.04.2e |
| 41(29H) | alu := tir; if n then goto 44; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0100 | 010.1100 | 30.00.04.2c |
| 42(2AH) | alu := ac; if n then goto 22; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0001 | 0001.0110 | 30.00.01.16 |
| 43(2BH) | goto 0; | 0 | 11 | 00 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0000 | 0000.0000 | 60.00.00.00 |
| 44(2CH) | alu := ac; if z then goto 0; | 0 | 10 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0001 | 0000.0000 | 50.00.01.00 |
| 45(2DH) | pc := band(ir, amask); goto 0; | 0 | 11 | 01 | 00 | 0 | 0 | 0 | 0 | 1 | 0000 | 1000 | 0011 | 0000.0000 | 68.10.83.00 |
| 46(2EH) | tir := lshift(tir); if n then goto 50; | 0 | 01 | 10 | 10 | 0 | 0 | 0 | 0 | 1 | 0100 | 0000 | 0100 | 0011.0010 | 34.14.04.32 |
| 47(2FH) | sp := sp + (-1); | 0 | 00 | 00 | 00 | 0 | 0 | 0 | 0 | 1 | 0010 | 0111 | 0010 | 0000.0000 | 00.12.72.00 |
| 48(30H) | mar := sp; mbr := pc; wr; | 0 | 00 | 10 | 00 | 1 | 1 | 0 | 1 | 0 | 0000 | 0010 | 0000 | 0000.0000 | 11.a0.20.00 |
| 49(31H) | pc := band(ir, amask); wr; goto 0; | 0 | 11 | 01 | 00 | 0 | 0 | 0 | 1 | 1 | 0000 | 1000 | 0011 | 0000.0000 | 68.30.83.00 |
| 50(32H) | tir := lshift(tir); if n then goto 65; | 0 | 01 | 10 | 10 | 0 | 0 | 0 | 0 | 1 | 0100 | 0000 | 0100 | 0100.0001 | 34.14.04.41 |
| 51(33H) | tir := lshift(tir); if n then goto 59; | 0 | 01 | 10 | 10 | 0 | 0 | 0 | 0 | 1 | 0100 | 0000 | 0100 | 0011.1011 | 34.14.04.3b |
| 52(34H) | alu := tir; if n then goto 56; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0100 | 0011.1000 | 30.00.04.38 |
| 53(35H) | mar := ac; rd; | 0 | 00 | 00 | 00 | 0 | 1 | 1 | 0 | 0 | 0000 | 0001 | 0000 | 0000.0000 | 00.c0.10.00 |
| 54(36H) | sp := sp + (-1); rd; | 0 | 00 | 00 | 00 | 0 | 0 | 1 | 0 | 1 | 0010 | 0111 | 0010 | 0000.0000 | 00.52.72.00 |
| 55(37H) | mar := sp; wr; goto 10; | 0 | 11 | 00 | 00 | 0 | 1 | 0 | 1 | 0 | 0000 | 0010 | 0000 | 0000.1010 | 60.a0.20.0a |
| 56(38H) | mar := sp; sp := sp + 1; rd; | 0 | 00 | 00 | 00 | 0 | 1 | 1 | 0 | 1 | 0010 | 0010 | 0110 | 0000.0000 | 00.d2.26.00 |
| 57(39H) | rd; | 0 | 00 | 00 | 00 | 0 | 0 | 1 | 0 | 0 | 0000 | 0000 | 0000 | 0000.0000 | 00.40.00.00 |
| 58(3AH) | mar := ac; wr; goto 10; | 0 | 11 | 00 | 00 | 0 | 1 | 0 | 1 | 0 | 0000 | 0010 | 0000 | 0000.1010 | 60.a0.20.0a |
| 59(3BH) | alu := tir; if n then goto 62; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0100 | 0011.1110 | 30.00.04.3e |
| 60(3CH) | sp := sp + (-1); | 0 | 00 | 00 | 00 | 0 | 0 | 0 | 0 | 1 | 0010 | 0111 | 0010 | 0000.0000 | 00.12.72.00 |
| 61(3DH) | mar := sp; mbr := ac; wr; goto 10; | 0 | 11 | 10 | 00 | 1 | 1 | 0 | 1 | 0 | 0000 | 0010 | 0001 | 0000.1010 | 71.a0.21.0a |
| 62(3EH) | mar := sp; sp := sp + 1; rd; | 0 | 00 | 00 | 00 | 0 | 1 | 1 | 0 | 1 | 0010 | 0010 | 0110 | 0000.0000 | 00.d2.26.00 |
| 63(3FH) | rd; | 0 | 00 | 00 | 00 | 0 | 0 | 1 | 0 | 0 | 0000 | 0000 | 0000 | 0000.0000 | 00.40.00.00 |
| 64(40H) | ac := mbr; goto 0; | 1 | 11 | 10 | 00 | 0 | 0 | 0 | 0 | 1 | 0001 | 0000 | 0000 | 0000.0000 | f0.11.00.00 |
| 65(41H) | tir := lshift(tir); if n then goto 73; | 0 | 01 | 10 | 10 | 0 | 0 | 0 | 0 | 1 | 0100 | 0000 | 0100 | 0100.1001 | 34.14.04.49 |
| 66(42H) | alu := tir; if n then goto 70; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0100 | 0100.0110 | 30.00.04.46 |
| 67(43H) | mar := sp; sp := sp + 1; rd; | 0 | 00 | 00 | 00 | 0 | 1 | 1 | 0 | 1 | 0010 | 0010 | 0110 | 0000.0000 | 00.d2.26.00 |
| 68(44H) | rd; | 0 | 00 | 00 | 00 | 0 | 0 | 1 | 0 | 0 | 0000 | 0000 | 0000 | 0000.0000 | 00.40.00.00 |
| 69(45H) | pc := mbr; goto 0; | 1 | 11 | 10 | 00 | 0 | 0 | 0 | 0 | 1 | 0000 | 0000 | 0000 | 0000.0000 | f0.10.00.00 |
| 70(46H) | a := ac; | 0 | 00 | 10 | 00 | 0 | 0 | 0 | 0 | 1 | 1010 | 0000 | 0001 | 0000.0000 | 10.1a.01.00 |
| 71(47H) | ac := sp; | 0 | 00 | 10 | 00 | 0 | 0 | 0 | 0 | 1 | 0001 | 0000 | 0010 | 0000.0000 | 10.11.02.00 |
| 72(48H) | sp := a; goto 0; | 0 | 11 | 10 | 00 | 0 | 0 | 0 | 0 | 1 | 0010 | 0000 | 1010 | 0000.0000 | 70.12.0a.00 |
| 73(49H) | alu := tir; if n then goto 76; | 0 | 01 | 10 | 00 | 0 | 0 | 0 | 0 | 0 | 0000 | 0000 | 0100 | 0100.1100 | 30.00.04.4c |
| 74(4AH) | a := band(ir, smask); | 0 | 00 | 01 | 00 | 0 | 0 | 0 | 0 | 1 | 1010 | 1001 | 0011 | 0000.0000 | 08.1a.93.00 |
| 75(4BH) | sp := sp + a; goto 0; | 0 | 11 | 00 | 00 | 0 | 0 | 0 | 0 | 1 | 0010 | 1010 | 0010 | 0000.0000 | 60.12.a2.00 |
| 76(4CH) | a := band(ir, smask); | 0 | 00 | 01 | 00 | 0 | 0 | 0 | 0 | 1 | 1010 | 1001 | 0011 | 0000.0000 | 08.1a.93.00 |
| 77(4DH) | a := inv(a); | 0 | 00 | 11 | 00 | 0 | 0 | 0 | 0 | 1 | 1010 | 0000 | 1010 | 0000.0000 | 18.1a.0a.00 |
| 78(4FH) | a := a + 1; goto 75; | 0 | 11 | 00 | 00 | 0 | 0 | 0 | 0 | 1 | 1010 | 0110 | 1010 | 0100.1011 | 60.1a.6a.4b |

**Chú ý:** (1) dấu '.' đã được "viết" thêm vào giữa các nhóm 4 bit nhị phân và các nhóm 2 số Hexa để cho dễ đọc; không "nạp" chúng vào Control-Store.

- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).

### 9.2.3. Mô phỏng sự hoạt động của Control Store

Với cách xây dựng Control Store thứ nhất, chỉ gồm 1 phần tử ROM lấy trong thư viện của Logisim, chúng ta chỉ có thể mô phỏng sự hoạt động của nó khi lắp ghép đầy đủ các thành phần của vi kiến trúc. Việc thực hành mô phỏng Control Store sẽ được thực hiện trong Bài tập mô phỏng số 12.

Với cách xây dựng Control Store thứ hai, chúng ta có thể mô phỏng việc đọc nội dung cũng như thực hiện import, export theo sơ đồ mạch điện được trình bày ở bài tập cuối chương dưới đây.

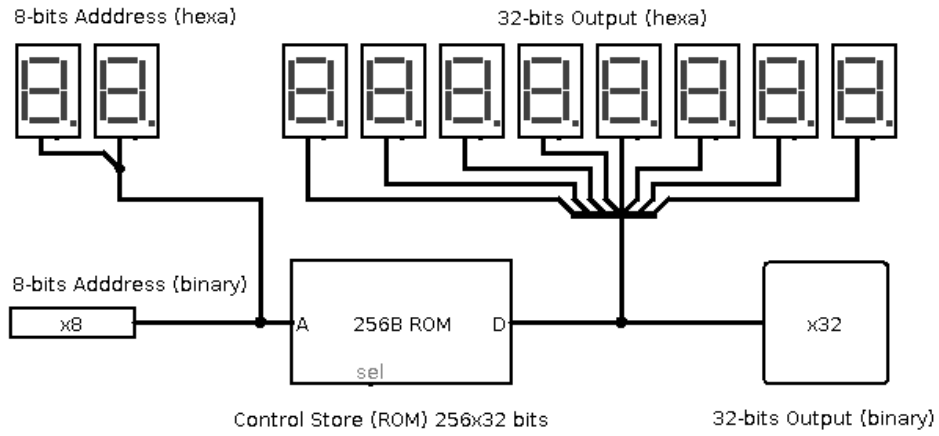
## 9.3 Bài tập

**Bài tập 9.1:** Tạo và mô phỏng hoạt động của mạch con Control-Store theo sơ đồ biểu diễn trên hình 9.5 dưới đây. Các yêu cầu cụ thể:

(1) Xây dựng mạch điện như hình vẽ;

(2) Soạn thảo nội dung cho ROM (chính là vi chương trình dưới dạng số Hexa), tập import, export;

(3) Thay đổi địa chỉ "8-bits address (binary)" và quan sát "32-bits Ouput (hexa)" xem có đúng như mình đã nạp vào không.




Hình 9.5 Control Store đượng bổ sung các phần tử  Hex display và Input, Output

# Bài số 10: Xây dựng MIR và Micro Seq của đơn vị điều khiển

## 10.1. Các yêu cầu và nội dung chính

Đơn vị điều khiển – CU (Control Unit) được giới thiệu khái quát trong mục "2.1 Kiến trúc chung của máy tính điện tử"; Tại chương 2, chúng ta đã biết rằng CU thực hiện việc điều khiển sự hoạt động của tất cả các đơn vị tạo nên hệ thống máy tính theo chương trình trong bộ nhớ chính.

CU được nghiên cứu sâu hơn trong chương 4, tại các mục "4.2.2 Vi chỉ thị - microinstruction", "4.2.3 Việc định thời vi chỉ thị", "4.2.4 Sự định trình tự các vi chỉ thị"... Trong sơ đồ khối đầy đủ của một vi kiến trúc ở hình 9.1 (Hình 4-09, GT KTMT), chúng ta thấy CU bao gồm 10 đơn vị sau: (1) Control Store, (2) MIR, (3) Micro Seq, (4) MPC, (5) Increment, (6) Mmux, (7) A Decoder, (8) B decoder, (9) C decoder và (10) đơn vị tạo tín hiệu đồng hồ - Clock.

Trong Bài tập mô phỏng số 10 này, chúng ta sẽ xây dựng và mô phỏng sự hoạt động của 2 đơn vị tiếp theo trong số 10 đơn vị cấu thành CU, đó là: MIR và Micro Seq.

- MIR: nhận vi chỉ thị đưa ra từ Control Store và duy trì nội dung trong thời gian vi chỉ thị này được thi hành.
- Micro Seq: sinh ra tín hiệu Mmux quyết định chọn vi chỉ thị tiếp theo được thi hành, căn cứ vào các tín hiệu trạng thái N, Z và tín hiệu điều khiển COND.

## 10.2. Xây dựng và mô phỏng sự hoạt động của MIR

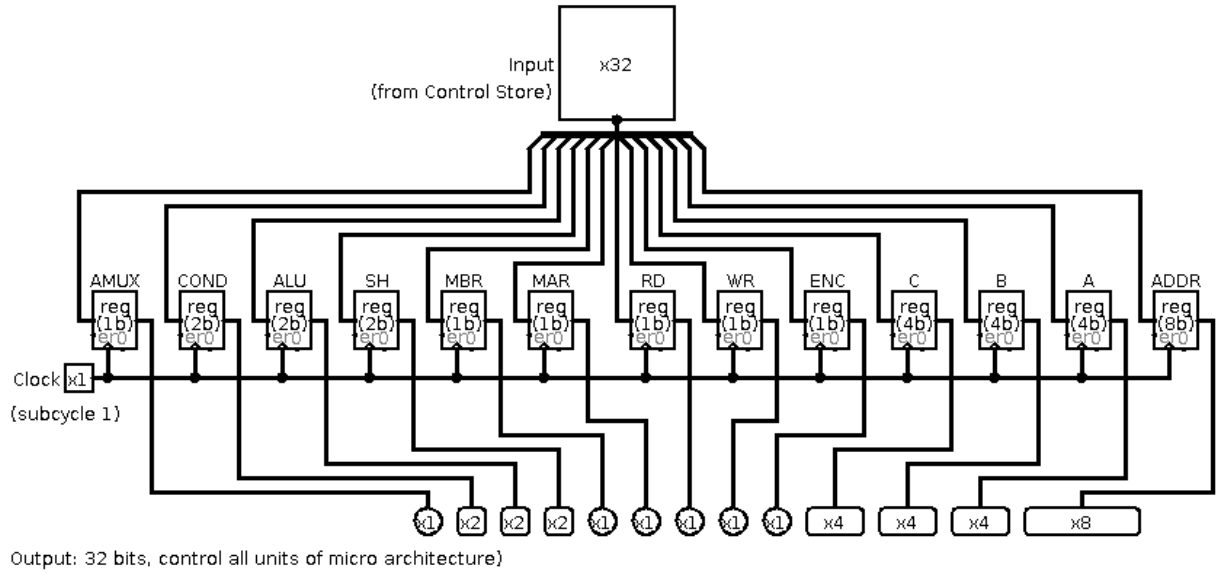
### 10.2.1 Phân tích

- Kích thước: 32 bits (chứa được 1 vi chỉ thị).
- Đầu vào: Có 1 đầu vào 32 bits nối với đầu ra của Control Store.
- Đầu ra: 32 bit được nhóm thành 13 nhóm tín hiệu, mỗi nhóm tín hiệu hoặc điều khiển một đơn vị thành phần định trước nào đó của đường dữ liệu (thí dụ: ALU, SH,...), hoặc đưa vào một đơn vị thành phần định trước nào đó thuộc CU để điều khiển một đơn vị thành phần nhất định thuộc CU (thí dụ: Micro Seq, Mmux, ) hoặc đường dữ liệu (thí dụ: các decoders).

### 10.2.2 Xây dựng thanh ghi MIR

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là MIR. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là MIR.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con MIR sẽ được lưu cùng với Micro-Architecture-SVxx.circ.

- Trong cửa sổ con “Canvas Window” của Logisim, xây dựng MIR như hình vẽ 10.1:




Hình 10.1 Thanh ghi MIR và các đầu vào, ra, điều khiển và đồng hồ

Chú ý: Các phần tử “reg” làm 16 thanh ghi trên có đầu vào “en” được Logisim đặt giá trị mặc định bằng 1 (active), vì vậy chúng ta không cần nối 16 đầu “en” của 16 thanh ghi lại với nhau và truyền vào đó giá trị “1”.

- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).

### 10.2.3 Mô phỏng hoạt động của thanh ghi MIR

Mô phỏng việc đưa dữ liệu từ “Input” (nối với Control-Store) vào MIR:

- Thiết lập giá trị cho phần tử “Input”, thí dụ bằng:

  “1111.0000.1111.0000.1111.0000.1111.0000”.

- Sử dụng phần tử “Poke tool” đảo giá trị của phần tử input “Clock” từ 0 lên 1, chúng ta sẽ thấy phần tử “Output” nhận giá trị bằng giá trị của phần tử “Input”.

## 10.3. Xây dựng và mô phỏng sự hoạt động của Micro Seq

### 10.3.1 Phân tích

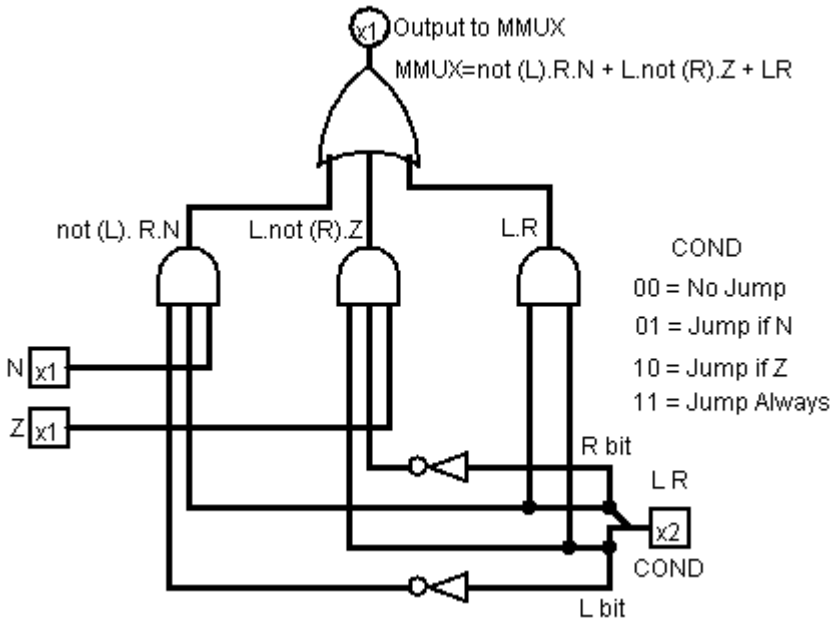
- Mmux = $\overline{L}RN + L\overline{R}Z + LR = RN + LZ + LR$
- Chúng ta sẽ xây dựng mạch sinh tín hiệu Mmux = $\overline{L}RN + L\overline{R}Z + LR$ cho dễ hiểu hơn.

### 10.3.2 Xây dựng mạch con Micro Seq

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là

Micro-Seq. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Micro-Seq.

- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Micro-Seq sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con "Canvas Window" của Logisim, xây dựng Micro-Seq như hình 10.2.
- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).




Hình 10.2 Đơn vị Micro-Seq

### 10.3.3 Mô phỏng hoạt động của Micro Seq

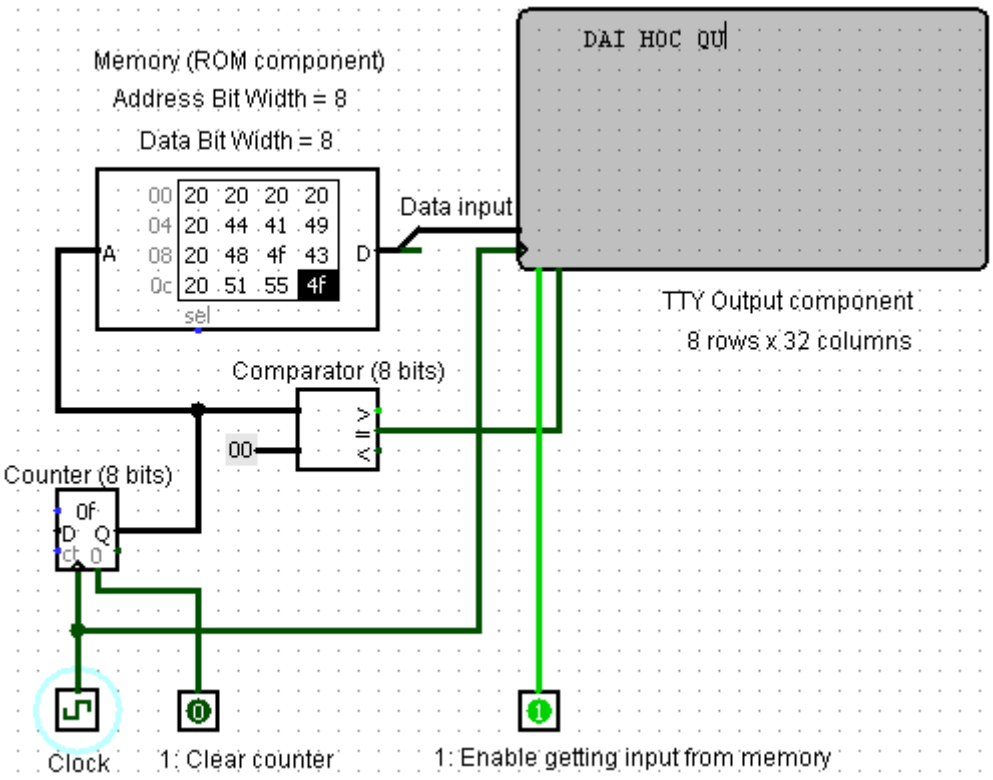
Mô phỏng hoạt động của mạch con Micro-Seq: Thiết lập giá trị cho phần tử input "COND" lần lượt bằng: 00, 01, 10 và 11, kiểm tra giá trị tại đầu ra MMUX phụ thuộc vào N và Z như thế nào:

- COND = 00: MMUX luôn bằng 0, không phụ thuộc vào N và Z.
- COND = 01: MMUX bằng 1 nếu N = 1 (N và Z không bao giờ đồng thời bằng 1).
- COND = 10: MMUX bằng 1 nếu Z = 1.
- COND = 11: MMUX luôn bằng 1, không phụ thuộc vào N và Z.

## 10.4. Bài tập

**Bài tập 10.1:** Hãy thiết kế bảng hiển thị điện tử kích thước 8 hàng x 32 cột và hiện dòng chữ "DAI HOC QUOC GIA HA NOI" ở chính giữa dòng trên cùng. Yêu cầu từng ký tự của dòng chữ trên được hiện lên lần lượt, sau khi hiển thị đến ký tự cuối cùng thì xóa toàn bộ và lặp lại quá trình trên.

Gợi ý: Có thể thiết kế "bảng hiển thị điện tử" theo sơ đồ trên hình 10.3 sau đây (sinh viên tự phân tích).




Hình 10.3 Thí dụ về một bảng hiện thị điện tử

Trong đó các phần tử (components) có thuộc tính như sau:

1. Phần tử TTY (TeleTYpe): Rows = 8, Columns = 32. Như vậy "màn hình" có thể hiển thị được tối đa là 8 x 32 = 256 ký tự. Hai giá trị này có thể thay đổi, thí dụ để "màn hình" hiển thị này giống màn hình máy tính ở chế độ Text, có thể chọn Rows = 25, Columns = 80. Cần chú ý: Đầu vào mã ký tự (Data) là 7 bit.
2. Phần tử ROM memory: Address Bit Width = 8 (dung lượng "con" ROM này sẽ là 2^8 = 256 Words); Data Bit Width = 8, do đó 1 word gồm 8 bit. Chú ý: Để nạp nội dung cho ROM (tức là nội dung hiển thị), hãy trỏ vào phần tử này rồi kích nút chuột phải, chọn "Edit contents..." và nhập vào các ô nhớ (byte) bắt đầu từ địa chỉ 00 các giá trị hexa sau: 20, 20, 20, 20, 20, 44, 41, 49, 20, 48, 4f, 43, 20, 51, 55, 4f, 43, 20, 47, 49, 41, 20, 48, 41, 20, 4e, 4f, 49.
3. Phần tử Comparator: Dùng để so sánh giá trị 8 bits đầu ra của bộ đếm (Counter) với hằng số 00h, khi 2 giá trị 8 bits này bằng nhau, tức là Counter đã đếm xong 1 vòng, từ 00 đến FFh và trở về 00, thì Comparator sẽ sinh ra giá trị 1 ("true") tại đầu ra "=", chúng ta đưa vào "Clear"

của phần tử TTY để xóa "màn hình" đồng thời chuyển "con trỏ màn hình" về góc trên bên trái.

4. Phần tử Counter: Giá trị tại đầu ra Q sẽ tăng thêm 1 mỗi khi đầu vào "Clock" có 1 xung (chuyển mức từ 0 lên 1). Logisim đặt giá trị ban đầu của Counter bằng 0. Đầu ra của Counter được chúng ta nối với đầu vào địa chỉ A của phần tử ROM, để lần lượt đánh địa chỉ các ô nhớ liên tiếp từ 00 đến FFh (256 ô nhớ). Ô nhớ được đánh địa chỉ sẽ truyền nội dung (1 byte) ra đầu ra dữ liệu D để truyền sang đầu vào dữ liệu của phần tử TTY. Chú ý: chúng ta sử dụng phần tử Splitter để nối 7 bit thấp của đầu ra D với đầu vào 7 bits của phần tử TTY.
5. Phần tử Clock: chúng ta có thể kích bằng "tay" hoặc bằng cách cho chạy tự động: (Simulate \ Ticks Enable), có thể thay đổi tốc độ hiển thị bằng cách thực hiện: Simulate \ Tick Frequency rồi chọn1 giá trị cụ thể. Nếu chúng ta chọn giá trị quá lớn, Logisim có thể không thực hiện được, khi đó chúng ta cần chọn lại một giá trị nhỏ hơn.
6. Phần tử Input (x 2): Chúng ta đã từng sử dụng nhiều lần ở các bài thực hành trước.
7. Phần tử Constant: Chúng ta đã từng sử dụng nhiều lần ở các bài thực hành trước.
8. Phần tử Splitter: Chúng ta đã từng sử dụng nhiều lần ở các bài thực hành trước.

# Bài số 11: Xây dựng các thành phần còn lại của đường điều khiển

Đơn vị điều khiển – CU (Control Unit) được giới thiệu khái quát trong mục "2.1 Kiến trúc chung của máy tính điện tử"; Tại chương 2 của GT KTMT, chúng ta đã biết rằng CU thực hiện việc điều khiển sự hoạt động của tất cả các đơn vị tạo nên hệ thống máy tính theo chương trình trong bộ nhớ chính.

CU được nghiên cứu sâu hơn trong chương 4, tại các mục "4.2.2 Vi chỉ thị - microinstruction", "4.2.3 Việc định thời vi chỉ thị", "4.2.4 Sự định trình tự các vi chỉ thị"... Trong sơ đồ khối đầy đủ của một vi kiến trúc ở hình 9.1 (Hình 4-09, GT KTMT), chúng ta thấy CU bao gồm 10 đơn vị sau: (1) Control Store, (2) MIR, (3) Micro Seq, (4) MPC, (5) Increment, (6) Mmux, (7) A Decoder, (8) B decoder, (9) C decoder và (10) đơn vị tạo tín hiệu đồng hồ - Clock. Trong Bài tập mô phỏng số 9, chúng ta đã xây dựng và mô phỏng sự hoạt động của Control Store; Trong Bài tập mô phỏng số 10, chúng ta đã xây dựng và mô phỏng sự hoạt động của MIR và Micro Seq.

## 11.1. Các yêu cầu và nội dung chính

Trong bài thực hành số 11 này chúng ta sẽ tạo ra và mô phỏng sự hoạt động của 7 đơn vị còn lại của CU, bao gồm các đơn vị sau:

- MPC: con trỏ vi chỉ thị.
- Increment: nhận input từ MPC rồi cộng thêm 1 và đưa ra đầu ra để truyền tới Mmux.
- Mmux: Bộ dồn kênh 2-to-1, độ rộng các đường vào/ra là 8 bits.
- A Decoder, B decoder và C decoder
  - A và B decoder là các bộ giải mã địa chỉ 4-to-16 thông thường.
  - C decoder: ngoài chức năng giải mã thông thường như A và B decoder, C decoder còn phải có 1 đặc điểm khác nữa: 1 trong 16 đầu ra được chọn chỉ được truyền mức logic 1 ra ngoài khi và chỉ khi ENC=1 đồng thời tín hiệu đồng hồ ở đầu ra subcycle 4 có mức 1.
- Clock: đơn vị tạo tín hiệu đồng hồ có 4 đầu ra cùng tần số do chúng ta chọn, nhưng lệch pha nhau 90 độ.

## 11.2. Xây dựng và mô phỏng hoạt động của thanh ghi MPC

### 11.2.1 Phân tích

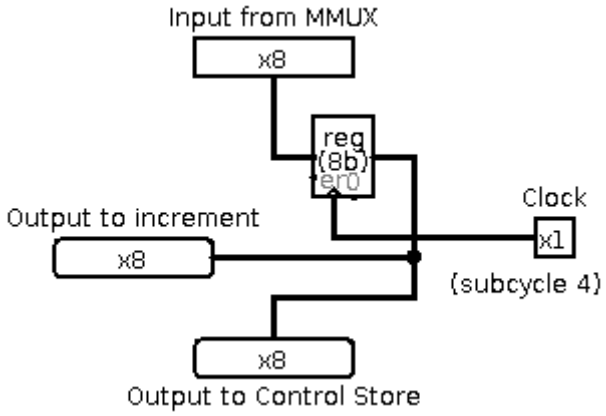
Như có thể thấy ở hình 9.1 Sơ đồ khối đầy đủ của một vi kiến trúc (Hình 4-09, GT KTMT) MPC có:

- 1 đầu vào dữ liệu 8 bits nối với đầu ra của đơn vị Mmux, đây chính là địa chỉ của vi chỉ thị trong Control Store cần được đưa ra MIR để thực hiện.
- 1 đầu ra dữ liệu 8 bits đưa vào Control Store.
- 1 đầu ra dữ liệu 8 bits đưa vào đơn vị Increment để làm tăng lên 1.

- 1 đầu vào điều khiển nối với đầu ra tín hiệu chu kỳ con thứ 4 (subcycle 4) của đơn vị Clock, chỉ khi tín hiệu này bằng 1 thì MPC mới chốt giá trị ở đầu vào (nghĩa là giá trị ở đầu ra nhận giá trị của đầu vào), còn khi tín hiệu này bằng 0 thì giá trị chứa trong MPC (và giá trị ở đầu ra) không thay đổi.

### 11.2.2 Xây dựng thanh ghi MPC

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là MPC. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là MPC.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con MPC sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con "Canvas Window" của Logisim, xây dựng MPC như hình vẽ 11.1.
- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).




Hình 11.1 Thanh ghi MPC và các đầu vào, ra và đồng hồ

### 11.2.3 Mô phỏng hoạt động của thanh ghi MPC

Mô phỏng việc đưa dữ liệu từ "Input from MMUX" vào MPC:

- Thiết lập giá trị nhị phân 8 bits cho phần tử "Input from MMUX", thí dụ bằng: 1111.0000.
- Sử dụng phần tử "Poke tool" đảo giá trị của phần tử input "subcycle 4" (Clock) từ 0 lên 1, chúng ta sẽ thấy phần tử "output to MIR" nhận giá trị bằng giá trị của phần tử "Input from MMUX".

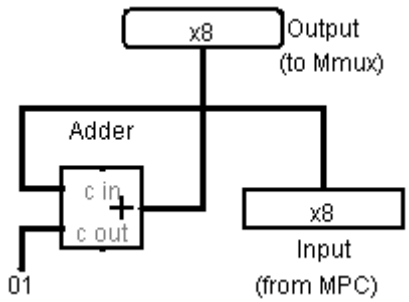
## 11.3. Xây dựng và mô phỏng hoạt động của đơn vị Increment

### 11.3.1 Phân tích

Với yêu cầu đã nêu trong mục 11.1 thì đây là bộ cộng, tóan hạng vào thứ nhất là dữ liệu ra từ MPC (chính là địa chỉ của chỉ thị đưa ra thi hành); tóan hạng vào thứ 2 là hằng +1. Kết quả của phép cộng sẽ đưa trở lại một trong hai đầu vào của Mmux.

### 11.3.2 Xây dựng mạch con Increment

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Increment. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Increment.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Increment sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con "Canvas Window" của Logisim, xây dựng Increment như hình 11.2.
- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).




Hình 11.2 Đơn vị Increment và các đầu vào, ra

### 11.3.3 Mô phỏng hoạt động của Increment

Mô phỏng việc thực hiện chức năng cộng 1 của Increment:

- Thiết lập giá trị nhị phân 8 bits cho phần tử "Input (from MPC)", thí dụ bằng: 1111.0000.
- Quan sát phần tử "Output (to Mmux)" chúng ta sẽ thấy giá trị vừa đưa vào tăng thêm 1, bằng 1111.0001.

## 11.4. Xây dựng và mô phỏng hoạt động của đơn vị Mmux

Đây là một bộ dồn kênh 2-to-1 thông thường, tương tự Amux mà chúng ta đã tìm hiểu trong bài thực hành số 8 (mục 8.4). Điều khác nhau duy nhất là với Mmux, các đường dữ liệu vào/ra là 8 bits chứ không phải 16 bits như với Amux.

Tại bài thực hành số 11 này chúng ta không tự xây dựng Mmux, đến bài thực hành số 12 chúng ta sẽ lấy một phần tử Multiplexer (thuộc nhóm Plexer trong thư viện của Logisim) làm Mmux.

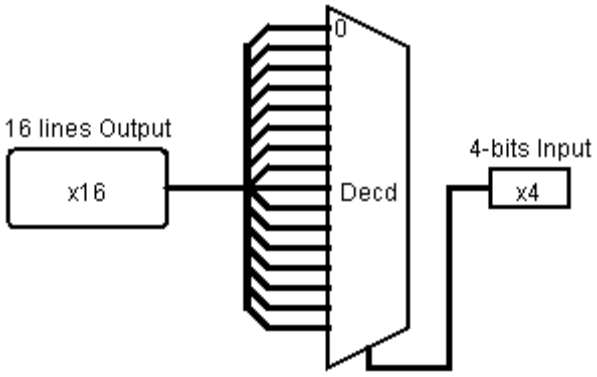
## 11.5. Xây dựng và mô phỏng hoạt động của decoder

### 11.5.1 Phân tích

- A và B decoders: đây là các decoder 4-to-16 thông thường.
- C decoders: theo các yêu cầu đã trình bày ở mục 11.1, mỗi một trong số 16 đầu ra của C decoder phải đi qua một bộ đệm 3 trạng thái (trong Logisim đó là phần tử controlled buffer) được điều khiển bởi tín hiệu là kết quả and lô-gic của ENC và "Clock subcycle 4".

### 11.5.2 Xây dựng mạch con làm A, B decoder và mô phỏng sự hoạt động

- Chúng ta có thể sử dụng phần tử Decoder trong nhóm Plexers trong thư viện của Logisim, hoặc chúng ta sử dụng các phần tử đó có bổ sung thêm một số phần tử khác để mạch con mà chúng ta xây dựng trông "gọn gàng" hơn, như sẽ thực hiện dưới đây.
- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Decoder-A&B. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Decoder-A&B.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Decoder-A&B sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con "Canvas Window" của Logisim, xây dựng Decoder-A&B như hình 11.3:




Hình 11.3 Decoder 4-to-16

**Chú ý:**

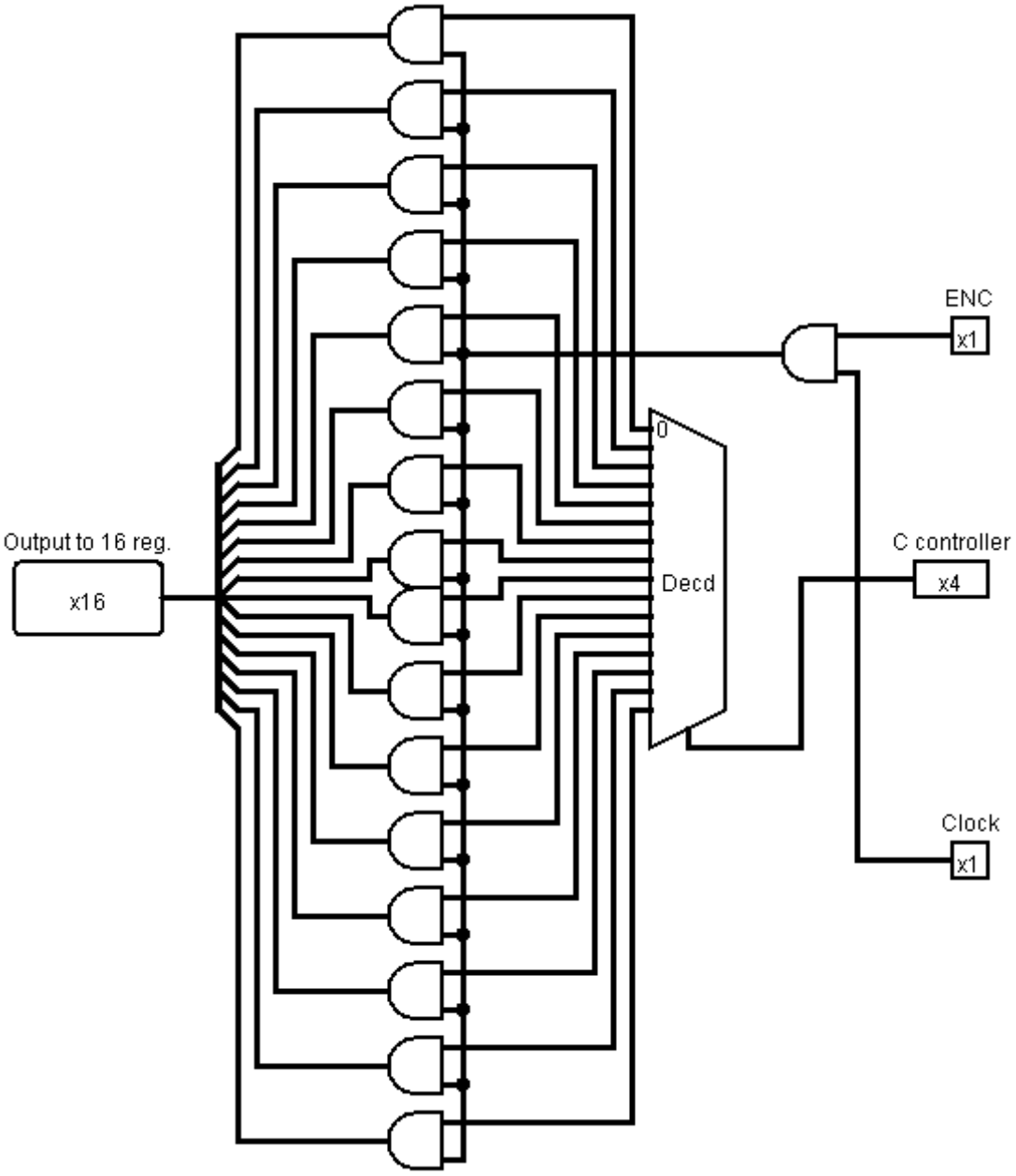
Việc chúng ta tạo ra phần tử Decoder-A&B chỉ nhằm mục đích thực hành mô phỏng. Nếu chúng ta dùng phần tử Decoder trong thư viện của Logisim, thì chúng ta không cần phải dùng thêm đến 3 phần tử, đó là:

- Phần tử input "4-bit input"
- Phần tử output "16 lines output"
- Phần tử Splitter để chập 16 dây ra từ decoder thành 1 bó dây cho gọn.

- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).
- Mô phỏng việc thực hiện chức năng của Decoder-A&B:
  - Thay đổi tăng dần giá trị của phần tử "4-bit input" từ 0000 đến 1111.
  - Quan sát phần tử "16 lines output" chúng ta sẽ thấy giá trị nhị phân 16 bit thay đổi, bit có giá trị 1 chuyển từ vị trí 0 đến vị trí 15, trong khi đó các bit còn lại bằng 0.

### 11.5.3 Xây dựng mạch con làm C decoder và mô phỏng sự hoạt động

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Decoder-C. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Decoder-C.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Decoder-C sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con "Canvas Window" của Logisim, xây dựng Decoder-C như hình 11.4.
- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).
- Mô phỏng việc thực hiện chức năng của Decoder-C: Thực hiện tương tự như khi mô phỏng Decoder-A&B, nhưng thực hiện 2 lần:
  - Lần thứ nhất thiết lập giá trị cho ENC và CLOCK sao cho ENC.CLOCK bằng 1.
    + Thay đổi tăng dần giá trị của phần tử "4-bit input" từ 0000 đến 1111.
    + Quan sát phần tử "16 lines output" chúng ta sẽ thấy giá trị nhị phân 16 bit thay đổi, bit có giá trị 1 chuyển từ vị trí 0 đến vị trí 15, trong khi đó các bit còn lại bằng 0.
  - Lần thứ hai thiết lập giá trị cho ENC và CLOCK sao cho ENC.CLOCK bằng 0.
    + Thay đổi tăng dần giá trị của phần tử "4-bit input" từ 0000 đến 1111.
    + Quan sát phần tử "16 lines output" chúng ta sẽ thấy giá trị nhị phân 16 bit không thay đổi, luôn bằng 0000.0000.0000.0000.




Hình 11.4 Đơn vị Decoder-C

## 11.6. Xây dựng và mô phỏng hoạt động của Clock

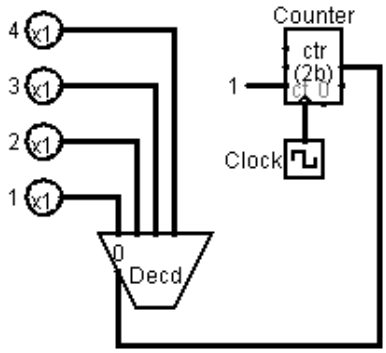
### 11.6.1 Phân tích

Đây là đơn vị không bị điều khiển bởi bất cứ đơn vị nào khác trong vi kiến trúc. Nhiệm vụ của nó là sinh ra 4 tín hiệu điện lệch pha nhau để điều khiển sự phối hợp công việc nhịp nhàng giữa các thành phần thuộc đường dữ liệu và thuộc đơn vị điều khiển CU.

Với vi kiến trúc được nghiên cứu trong môn học Kiến trúc máy tính, trừ các chỉ thị Read và Write, các chỉ thị khác được thi hành trong thời gian như nhau và được gọi là 1 chu kỳ đường dữ liệu. Như vậy, các tín hiệu đồng hồ được coi là lệch pha nhau $90^0$ (vấn đề được giải thích chi tiết tại mục “4.2.3 Việc định thời vi chỉ thị”, GT KTMT).

## 11.6.2 Xây dựng mạch con làm đơn vị Clock

- Chạy Logisim rồi mở file Micro-Architecture-Svxx.circ, vào menu Project \ Add Circuit, Logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Clock-4-subcycles. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, Logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Clock-4-subcycles.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Clock-4-subcycles sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con "Canvas Window" của Logisim, xây dựng Clock-4-subcycles như hình vẽ sau đây. Vì trong thư viện của Logisim chỉ có 1 loại phần tử clock 1 pha, cho nên chúng ta bắt buộc phải sử dụng thêm một số phần tử khác nữa để tạo 4 tín hiệu đồng hồ có cùng tần số nhưng lệch pha nhau $90^0$. Các phần tử được sử dụng để xây dựng mạch con này bao gồm:
  - Clock: lấy trong thư viện của Logisim. Số thuộc tính của nó có thể thay đổi được là 5 (Facing, High Duration, Low Duration, Label, Label Location, Label Font), ý nghĩa các thuộc tính rất dễ hiểu. Ngoài ra khi mô phỏng hoạt động của clock, chúng ta có thể thay đổi được:
    + Tần số: từ 0.25 Hz đến 4096 Hz (vào Menu: Simulate \ Tick Frequency).
    + Cách chạy đồng hồ: 1/ Tick One (để debug); 2/ Tick Enable (để chạy bình thường).




Hình 11.5 Đơn vị Clock phát tín hiệu 4 chu kỳ con

  - Counter: phần tử đếm có trong thư viện. Counter có các thuộc tính sau:
    + Data bits (1..32): độ lớn của số mà Counter sẽ đếm. Chúng ta cần đặt bằng 2 để Counter đếm từ 0 đến 3.
    + Maximum value: giới hạn số cao nhất mà Counter đếm, phải đặt không lớn hơn giá trị được xác định bởi số chứa được trong "Data bits".
    + Action on Overflow: 1/ Wrap around (đếm vòng lại từ số bé nhất); 2/ Stay at value (dừng đếm) ...
    + Trigger: 1/ Rising Edge (kích hoạt việc đếm bởi sườn dương); 2/ Falling Edge (kích hoạt việc đếm bằng sườn âm).

+ Label: nhãn do chúng ta đặt.
+ Label Font: kiểu và cỡ font chữ của Label.

Trên hình vẽ, Counter có tất cả 7 chân vào/ra. Trong "Canvas Window", khi chúng ta di con trỏ đến các chân này, Logisim sẽ hiện một dòng giải thích:

+ "^": Clock: value may update on trigger. Khi tín hiệu (xung đồng hồ) đặt vào đây thay đổi từ mức thấp lên mức cao, sẽ làm thay đổi giá trị chứa trong counter 1 đơn vị. Sự thay đổi có thể là tăng hoặc giảm phụ thuộc vào một số tín hiệu điều khiển khác, sẽ được mô tả dưới đây.
+ "0": Clear: clear, when 1 resets to 0 asynchronously. Giá trị mặc định ở đầu vào này là 1, chúng ta có thể để trống chân này.
+ "ct": Count, when 1, counter increments (or decrement if load =1). Nếu đưa giá trị 1 vào đây, counter sẽ đếm tiến.
+ "D": value to load into counter. Đây là đầu vào cho giá trị ban đầu (nhiều bit) mà chúng ta muốn nạp cho counter.
+ "Q": Output: current value of counter. Đây là đầu ra cho giá trị hiện thời của counter.
+ Đầu vào trên cùng bên trái (không có ký hiệu): Load: when 1, loads from data input (if Count =0) or decrements.

– Decd (Decoder): đây là phần tử giải mã thông thường, có sẵn trong thư viện.
– 4 phần tử output được ghi nhãn 1, 2, 3, 4 tương ứng với 4 đầu ra truyền tín hiệu của 4 chu kỳ con.

- Khi xây dựng xong mạch con hoặc sau mỗi khoảng thời gian làm việc nhất định, hoặc khi muốn kết thúc Logisim, hãy lưu (save) tất cả vào file (Micro-Architecture-Svxx.circ).

### 11.6.3 Mô phỏng sự hoạt động của Clock

Hãy chọn "Tick Enable" hoặc chọn "Tick Once" nhiều lần, chúng ta sẽ nhìn thấy nội dung phần tử counter thay đổi tăng dần từ 0 đến 3 và vòng lại giá trị 0.

Có thể thay đổi tần số đồng hồ để thay đổi tốc độ đếm (Simulate \ Tick Frequency).

## 11.7. Bài tập

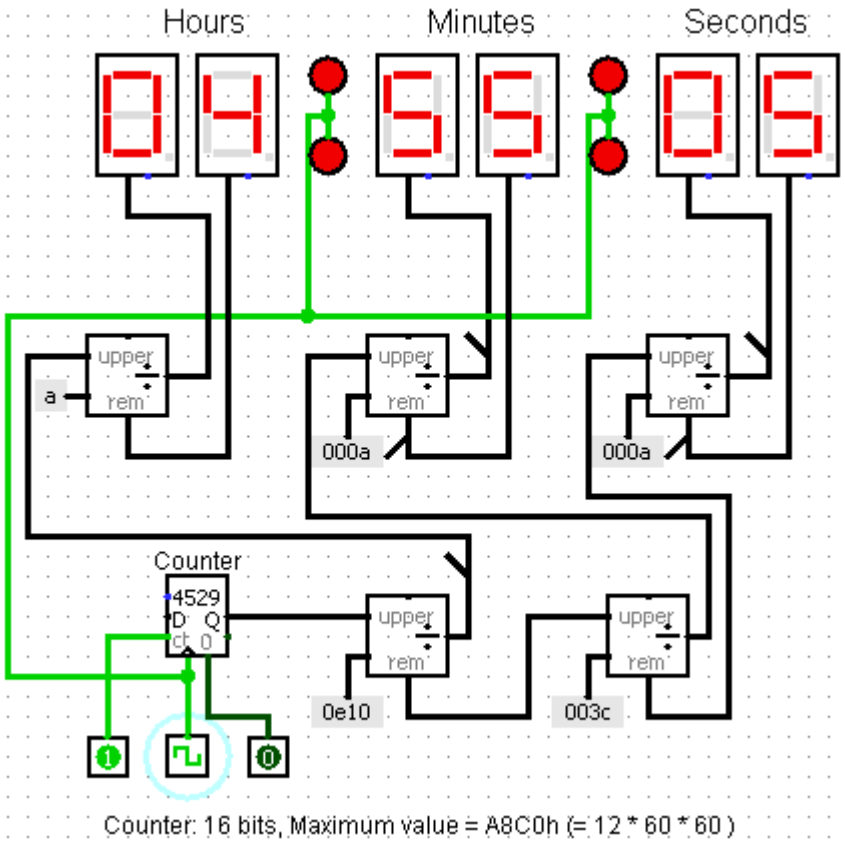
**Bài tập 11.1:** Thiết kế một đồng hồ điện tử hiện số với các yêu cầu sau:

1. Thời gian được hiển thị dưới dạng: hh : mm : ss, trong đó:

- hh là số chỉ giờ (hour), tăng từ 0 đến 11 rồi lại trở về đếm từ 0.
- mm là số chỉ phút (minute), tăng từ 0 đến 59 rồi lại trở về đếm từ 0.
- ss là số chỉ giây (second), tăng từ 0 đến 59 rồi lại trở về đếm từ 0.
- Giữa các số chỉ thời gian nói trên có dấu ":", nhấp nháy 1 giây 1 lần.

2. Mỗi chữ số được hiển thị bằng 1 phần tử "Hex Digit Display" trong thuộc nhóm Input/Ouput trong thư viện của Logisim.

**Gợi ý:** Có thể thiết kế đồng hồ theo sơ đồ ở hình 11.6 dưới đây. Các phần tử được sử dụng đều có trong thư viện của Logisim, mô tả về chức năng hoạt động của chúng cũng có thể tra cứu trong "Help".




Hình 11.6 Một thí dụ về mạch điện của đồng hồ điện tử

# Bài số 12: Xây dựng vi kiến trúc đầy đủ

## 12.1. Các yêu cầu và nội dung chính

Trong bài thực hành này, chúng ta cần thực hiện được 3 việc chính sau:

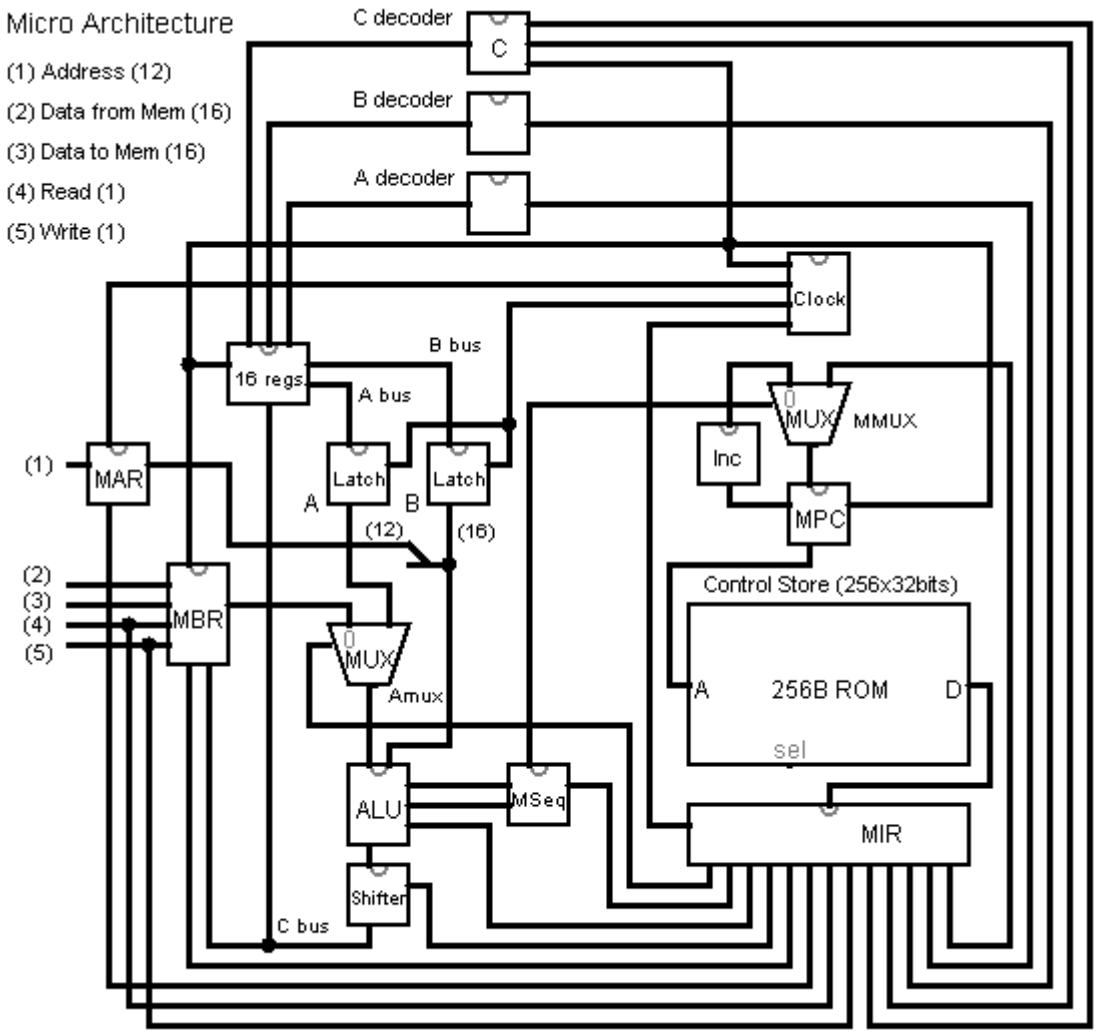
1. Lắp ghép các đơn vị chức năng để tạo thành vi kiến trúc đầy đủ, như trên hình 12.1 (Hình 4-09 trong GT KTMT, có trong Bài tập mô phỏng số 9).
2. Bổ sung thêm bộ nhớ chính (RAM) và ghép nối với vi kiến trúc để tạo nên một "máy tính" chạy được chương trình, như trên hình 12.2.
3. Kiểm tra "máy tính" xem nó có được lắp ghép đúng đắn hay không, kiểm tra vi chương trình dưới dạng số Hexa xem chúng ta đã dịch từ dạng MAL (Micro Assembly Language) sang có chính xác hay không bằng cách cho thực hiện một chương trình viết bằng tập chỉ thị của máy MAC-1 (có 23 chỉ thị, tương tự các chỉ thị của ngôn ngữ Assembly).

## 12.2. Lắp ghép vi kiến trúc đầy đủ

Khi bắt đầu chạy Logisim, chúng ta vào thẳng ngay cửa sổ con "Canvas Window" của mạch chính là Main. Nếu chúng ta đang làm việc với một mạch con khác, hãy kích vào tên Main. Trong cửa sổ con "Canvas Window" của Logisim, xây dựng vi kiến trúc đầy đủ như trên hình vẽ 12.1.

Kiểm tra vi kiến trúc:

- Sau khi đã xây dựng một công trình "đồ sộ" và "tinh vi" như thế này, việc kiểm tra xem các đơn vị cấu thành có thực hiện đúng chức năng theo thiết kế hay không, việc lắp ghép chúng có sai sót gì không thực sự là một việc đòi hỏi sự tỉ mỉ và không dễ dàng.
- Để việc kiểm tra được thuận lợi, trước khi kiểm tra chúng ta sẽ bổ sung thêm bộ nhớ chính (RAM) cùng với các phần tử để ghép nối RAM với vi kiến trúc và một số phần tử Output để chúng ta có thể quan sát sự hoạt động của vi kiến trúc được thuận lợi. Sinh viên khi thực hành hãy dựa vào hình 12.2 dưới đây để bổ sung các phần tử nói trên vào vi kiến trúc của mình.




Hình 12.1 Vi kiến trúc đầy đủ

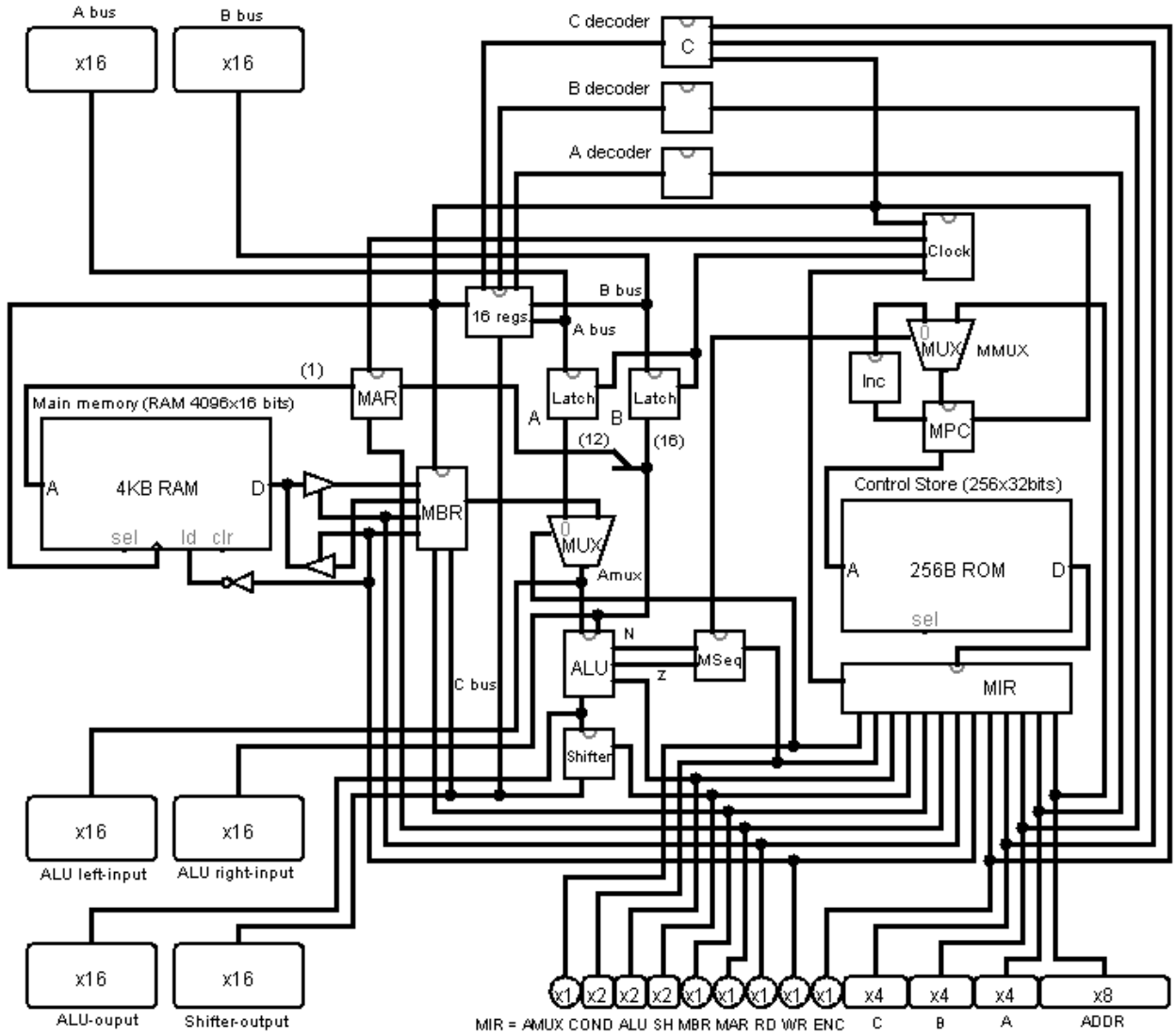
## 12.3. Bổ sung bộ nhớ chính và lắp ghép với vi kiến trúc

Bộ nhớ chính (Main memory) đương nhiên là RAM, chúng ta sẽ sử dụng một phần tử RAM có trong thư viện của Logisim. Trong mục "4.3 Thí dụ về một kiến trúc mức máy thông thường", chúng ta đã giả sử bộ nhớ chính có dung lượng 4 K words, kích thước word là 16 bit. Như vậy phần tử nhớ RAM mà chúng ta chọn trong thư viện của Logisim làm bộ nhớ chính cần được thiết lập các thuộc tính như sau:

- Address Bit Width: 12 bit (để có thể chọn 1 trong 2^12 = 4 K words)
- Data Bit Width: 16 bit (kích thước 1 vĩ chỉ thị)
- Data Interface: One synchronous load/store port (Có một cổng để đọc/ghi dữ liệu kiểu đồng bộ, nghĩa là nối với bus đồng bộ)

Nội dung bộ nhớ chính: Chứa chương trình ở mức "Ngôn ngữ máy thông thường" của người sử dụng. Trong tài liệu này, chúng ta sẽ tìm hiểu một thí dụ, trong đó chúng ta sẽ nạp vào bộ nhớ chính các chương trình như vậy để kiểm tra hệ thống đã xây dựng.

Chúng ta bổ sung thêm bộ nhớ chính (RAM) và ghép nối với vi kiến trúc để tạo nên một "máy tính" chạy được chương trình ở mức "Ngôn ngữ máy thông thường", như trên hình 12.2.




Hình 12.2 Vi kiến trúc đầy đủ được bổ sung thêm bộ nhớ RAM 4 KB

## 12.4. Kiểm tra hệ thống máy tính

Để kiểm tra hệ thống máy tính, cách tốt nhất là cho chạy thử một hoặc một số chương trình, sao cho tất cả 23 vĩ chỉ thị (xem mục "4.3.3 Tập vĩ chỉ thị", trong giáo trình "Kiến trúc máy tính") đều được lấy về (Fetch), giải mã (Decode) và thực hiện (Execute). Điều đó cũng có nghĩa là toàn bộ các vi chỉ thị của vi chương trình với chức năng thông dịch mà chúng ta đã viết (xem mục "4.4.2 Thí dụ về một vi chương trình") đều được thực hiện.

Nếu tất cả các chương trình đều thực hiện đúng các yêu cầu của thuật tóan mà nó thể hiện, thì có thể kết luận rằng hệ thống máy tính của chúng ta đã được thiết kế và thi công đúng đắn. Hệ thống máy tính nói ở đây bao gồm 2 thành phần: 1/ Phần cứng: đó là vi kiến trúc như ở hình vẽ trên; 2/ Phần mềm hệ thống: đó là vi chương trình thực hiện chức năng thông dịch mà chúng ta đã viết và nạp vào Control Store.

Để kiểm tra hệ thống máy tính, chúng ta sẽ viết và cho chạy một chương trình, chương trình này sẽ sử dụng một số chỉ thị trong tập 23 vĩ chỉ thị. Sau khi viết xong, chúng ta sẽ tự tay dịch nó sang dạng số nhị phân và dạng số Hexa, sau đó nạp (load) nó vào trong bộ nhớ chính (main memory) và cho thi hành. Trong quá trình chương trình đang được thi hành, chúng ta có thể theo dõi tất cả các thành phần của hệ thống máy tính kể cả các thành phần mới bổ sung. Khi chương trình được thi hành xong, chúng ta sẽ kiểm tra kết quả (nằm trong bộ nhớ chính) xem có đúng không.

**Chương trình kiểm tra đầu tiên và đơn giản nhất: test01-addition.txt**

- Mô tả chương trình:
  - Input: Cho 2 số 1234h và 1010h
  - Output: Tổng của chúng
- Phân tích việc viết chương trình bằng vĩ chỉ thị:

  Với một bài tóan cộng đơn giản như thế này, không có gì đáng để nói về thuật tóan giải quyết, chúng ta có thể nhẩm ra ngay kết quả: 1234h + 1010h = 2244h.

  Tuy nhiên để lập trình cho hệ thống máy tính đơn giản của chúng ta, có lẽ chỉ có một kiểu chương trình thích hợp nhất, đó là kiểu chương trình dạng ".com" theo cách gọi của những người lập trình Hợp ngữ (Assembly language) trong môi trường hệ điều hành DOS của hãng Microsoft. Các chương trình dạng ".com" có các đặc điểm chính sau:

  1. Các chỉ thị của chương trình và dữ liệu của chương trình khi được nạp vào bộ nhớ chính sẽ chiếm một vùng các ô nhớ liên tiếp.
  2. Ô nhớ đầu tiên luôn là một lệnh nhảy vô điều kiện (JMP) qua vùng các ô nhớ chứa dữ liệu nằm ngay sau nó. Trong vùng chứa dữ liệu, một số ô nhớ sẽ chứa các tóan hạng cho các chỉ thị của chương trình (giống như các ô nhớ tương ứng với các biến và các hằng của các chương trình khả thi được dịch từ chương trình viết bằng C++ hay Pascal); một số ô nhớ khác có thể chứa kết quả.

  Ngoài vấn đề cần viết chương trình dạng ".com", còn có một vấn đề khác, đó là chương trình của chúng ta sẽ kết thúc như thế nào? Hay nói cách khác sau khi thực hiện xong, máy tính của chúng ta có dừng được không và dừng như thế nào. Người lập trình trên các máy tính có hệ điều hành nói chung không cần quan tâm đến vấn đề này.

  Trong phạm vi môn học này, để cho đơn giản, chúng ta sẽ "dừng" chương trình của mình bằng cách viết một lệnh nhảy vô điều kiện đến chính nó, nghĩa là nhảy đến ô nhớ chứa chính lệnh nhảy này. Lệnh nhảy như vậy chính là một vòng lặp vô hạn, dùng để "dừng" chương trình; Có thể so sánh cách dừng này với hình ảnh một người chạy dậm chân tại chỗ!
- Viết chương trình:
  - Ô nhớ có địa chỉ 000 chứa lệnh (vĩ chỉ thị) nhảy qua vùng dữ liệu.
  - Ô nhớ có địa chỉ 001 chứa dữ liệu là con số 1234h.
  - Ô nhớ có địa chỉ 002 chứa dữ liệu là con số 1010h.

- Ô nhớ có địa chỉ 003 chứa dữ liệu là kết quả của phép cộng 2 số nói trên. Ban đầu chúng ta thiết lập cho nó giá trị 0, giống như với nhiều ngôn ngữ lập trình bậc cao, các biến chưa được khởi tạo thường được cho bằng 0.
- Ô nhớ có địa chỉ 004 chứa chỉ thị LODD với địa chỉ tóan hạng bộ nhớ là 001.
- Ô nhớ có địa chỉ 005 chứa chỉ thị ADDD với địa chỉ tóan hạng bộ nhớ là 002.
- Ô nhớ có địa chỉ 006 chứa chỉ thị STOD với địa chỉ tóan hạng bộ nhớ là 003.
- Ô nhớ có địa chỉ 007 chứa chỉ thị JMP với địa chỉ nhảy đến là 007.

Chương trình của chúng ta được trình bày trong bảng dưới đây; ý nghĩa của các cột như sau:

- Cột đầu tiên (Memory address) là địa chỉ bộ nhớ chính cần nạp chương trình vào.
- Cột thứ 2 (Content) là chỉ thị viết dưới dạng số Hexa.
- Cột thứ 3 (MAC-1 instruction) là ký hiệu vĩ chỉ thị tương ứng và địa chỉ tóan hạng X.
- Cột ngoài cùng bên phải (Equivalent Micro Instruction) là nhóm các vi chỉ thị tương ứng với vĩ chỉ thị ghi ở cột 2.

| Main Memory address | Content (Hex) | MAC-1 instruction | Equivalent Micro Instruction |
|---|---|---|---|
| 000 | 6004 | JMP ;X=004 | 00: mar := pc; rd;<br>01: pc := pc+1; rd;<br>02: ir := mbr; if n then goto 28;<br>03: tir :=lshift(ir+ir); if n then goto 19;<br>19: tir:=lshift(tir); if n then goto 25;<br>25: alu := tir; if n then goto 27; {0110 or 0111?}<br>26: pc := band(ir, amask); goto 0; {pc := x}<br>00: mar := pc; rd; |
| 001 | 1234 | | |
| 002 | 1010 | | |
| 003 | 0000 | | |
| 004 | 0001 | LODD ;X=001 | 00: mar := pc; rd;<br>01: pc := pc+1; rd;<br>02: ir := mbr; if n then goto 28;<br>03: tir :=lshift(ir+ir); if n then goto 19;<br>04: tir := lshift(tir); if n then goto 11;<br>05: alu := tir; if n then goto 9;<br>06: mar := ir; rd; {mar := operand add, and read}<br>07: rd; {keep on reading}<br>08: ac := mbr; goto 0; {ac := Operand}<br>00: mar := pc; rd; |
| 005 | 2002 | ADDD ;X=002 | 01: pc := pc+1; rd;<br>02: ir := mbr; if n then goto 28;<br>03: tir :=lshift(ir+ir); if n then goto 19;<br>04: tir := lshift(tir); if n then goto 11;<br>11: alu := tir; if n then goto 15; {0010 or 0011?}<br>12: mar := ir; rd; {mar := operand add, and read}<br>13: rd; {keep on reading}<br>14: ac := mbr + ac; goto 0; {going to fetch a next instruction}<br>00: mar := pc; rd; |

| Main Memory address | Content (Hex) | MAC-1 instruction | Equivalent Micro Instruction |
|---|---|---|---|
| 006 | 1003 | STOD ;X=003 | 00: mar := pc; rd;<br>01: pc := pc+1; rd;<br>02: ir := mbr; if n then goto 28;<br>03: tir :=lshift(ir+ir); if n then goto 19;<br>04: tir := lshift(tir); if n then goto 11;<br>05: alu := tir; if n then goto 9;<br>9: mar := ir; mbr := ac; wr; {mar := operand add, and write}<br>10: wr; goto 0; {keep on writing, going to fetch a next instruction}<br>00: mar := pc; rd; |
| 007 | 6007 | JMP ;X=007 | 00: mar := pc; rd;<br>01: pc := pc+1; rd;<br>02: ir := mbr; if n then goto 28;<br>03: tir :=lshift(ir+ir); if n then goto 19;<br>19: tir:=lshift(tir); if n then goto 25;<br>25: alu := tir; if n then goto 27; {0110 or 0111?}<br>26: pc := band(ir, amask); goto 0; {0110 = JUMP}<br>00: mar := pc; rd; |

- Xác định tập vi chỉ thị sẽ thực hiện chương trình bằng vĩ chỉ thị:

  Qua việc phân tích ở trên, có thể thấy rằng để thực hiện chương trình của chúng ta, các vi chỉ thị sau đây trong Control Store sẽ được thi hành. Trong số đó các chỉ thị ở dòng số 0, 1, 2, 3, 4 và 5 sẽ được thi hành nhiều lần vì chúng thuộc các bước "Fetch" và "Decode" của việc thực hiện mỗi dòng lệnh trong chương trình của chúng ta.

  0: mar := pc; rd; {main loop. Fetch an Instruction}
  1: pc := pc+1; rd; {increment pc, and keep on reading}
  2: ir := mbr; if n then goto 28; {save, if opcode = 1xxx.12x then goto 28}
  3: tir :=lshift(ir+ir); if n then goto 19; {if opcode = 01xx.12x then goto 19 else next line}
  4: tir := lshift(tir); if n then goto 11; {000x or 001x?}
  5: alu := tir; if n then goto 9; {0000 or 0001?}

  19: tir:=lshift(tir); if n then goto 25;
  25: alu := tir; if n then goto 27; {0110 or 0111?}
  26: pc := band(ir, amask); goto 0; {pc := x}

  6: mar := ir; rd; {mar := operand add, and read}
  7: rd; {keep on reading}
  8: ac := mbr; goto 0; {ac := Operand}

  11: alu := tir; if n then goto 15; {0010 or 0011?}
  12: mar := ir; rd; {mar := operand add, and read}
  13: rd; {keep on reading}
  14: ac := mbr + ac; goto 0; {going to fetch a next instruction}

  9: mar := ir; mbr := ac; wr; {mar := operand add, and write}

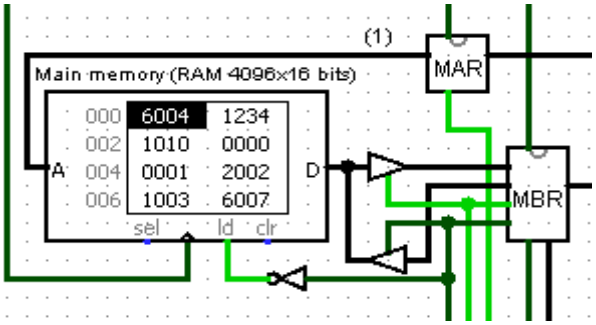
10: wr; goto 0; {keep on writing, going to fetch a next instruction}

- Nạp chương trình vào bộ nhớ chính:

  Có 2 cách:

  1. Trong cửa sổ Main của Logisim, trỏ vào phần tử Main Memory (RAM) rồi kích nút chuột phải, chọn "Edit contents...", Logisim sẽ hiện lên cửa sổ "Hex Editor" để cho ta soạn thảo nội dung từng ô nhớ bằng số Hexa. Để đóng cửa sổ, kích vào nút "Close Windows". Trong khi đang soạn thảo hoặc trước khi kết thúc soạn thảo nên thực hiện thao tác "Save" để ghi nội dung Main Memory vào một file. Trong chương trình kiểm tra thứ nhất này, chúng ta đặt tên file là test01-addition.txt.
  2. Dùng một chương trình soạn thảo văn bản tạo ra một file kiểu Plain Text, đặt tên file là: test01-addition.txt. Nội dung file gồm các word ở cột 2 – "Content (Hex)" của bảng trên, các word cách nhau bởi ký tự trống (ký tự cách - space). Trong cửa sổ Main của Logisim, trỏ vào phần tử Main Memory (RAM) rồi kích nút chuột phải, chọn "Load Image...", Logisim sẽ hiện lên cửa sổ để ta tìm file cần load.

  Hình ảnh của bộ nhớ Main Memory sau khi nạp chương trình như sau:




Hình 12.3 Hình ảnh bộ nhớ chính sau khi nạp chương trình

- Cách cho chạy chương trình:

  Sau khi đã nạp chương trình vào bộ nhớ, chúng ta cần chọn: Simulate \ Simulation Enabled để có thể chạy mô phỏng chương trình. Chúng ta có thể cho chạy chương trình theo một trong hai cách:

  1. Chạy liền một mạch: Simulate \ Tick Enabled (hoặc CTRL_K). Chỉ nên chạy một chương trình theo cách này khi đã từng chạy chương trình này thành công.
  2. Chạy từng bước: Simulate \ Tick Once (hoặc CTRL_T). Cách chạy này giúp chúng ta dễ dàng gỡ rối (debug) chương trình cũng như có thể phát hiện các sai sót về thiết kế của các đơn vị tạo nên hệ thống cũng như việc ghép nối các đơn vị. Theo cách này, gõ CTRL_T 2 lần sẽ ứng với 1 chu kỳ con đồng hồ và toàn bộ hệ thống chỉ thực hiện công việc trong 1 chu kỳ con của đơn vị đồng hồ rồi dừng lại. Để thực hiện 1 vi chỉ thị cần gõ CTRL_T 8 lần.

- Giám sát việc chạy chương trình:

Cần cho chạy chương trình theo từng bước, mỗi khi chương trình dừng lại, chúng ta cần xem nội dung các thanh ghi, các ô nhớ... mà chúng ta biết chắc rằng vi chỉ thị làm thay đổi, hoặc không làm thay đổi so với ở chu kỳ con trước đó, để xem xem có đúng như vậy hay không.

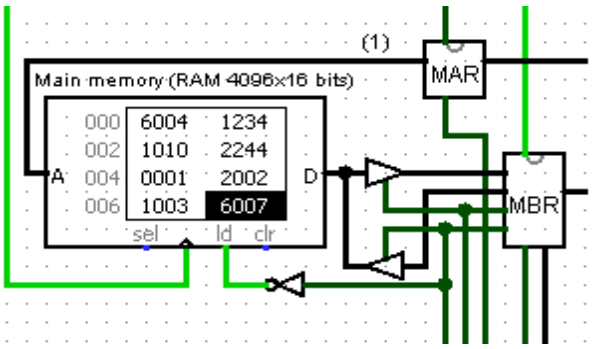
Thí dụ:

- Ở chu con đầu tiên (sau khi nạp chương trình vào Main memory): mar = 1, rd = 1, z = 1 và đầu ra chu kỳ con thứ nhất (của đơn vị đồng hồ) = 1.
- Ở chu kỳ con thứ 2: mar = 1, rd = 1, z = 1 và đầu ra chu kỳ con thứ hai = 1.
- Ở chu kỳ con thứ 3: mar = 1, rd = 1, z = 1, nội dung thanh ghi MAR=000 (địa chỉ chỉ thị đầu tiên trong Main Memory) và đầu ra chu kỳ con thứ ba = 1.
- Ở chu kỳ con thứ 4: mar = 1, rd = 1, z = 1, nội dung thanh ghi MAR=000, MBR=000 và đầu ra chu kỳ con thứ tư = 1.
- Ở chu kỳ con thứ 1 của vi chỉ thị kế tiếp: mar = 0, rd = 1, z = 1, ALU = 00 (phép cộng), ENC =1, A = 0000 (chọn thanh ghi số 0 là pc để đưa nội dung ra bus A), B = 0110 (chọn thanh ghi số 6 là +1 để đưa nội dung ra bus B), C = 0000 (chọn thanh ghi số 0 là pc để chứa kết quả trên bus C)
- nội dung thanh ghi MAR=000, MBR=000 và đầu ra chu kỳ con thứ tư = 1.
- v.v.

Đây là một công việc khó, đòi hỏi sự nắm vững kiến thức về toàn bộ hệ thống, tập trung tư tưởng và rèn luyện để tích lũy kinh nghiệm.

Các bạn sinh viên cần dành đủ thời gian để giám sát việc thực hiện chương trình theo tất cả các bước, từ bước đầu tiên cho đến khi nhìn thấy tại ô nhớ 003 có giá trị bằng 2244 (kết quả của phép cộng). Sau khi thấy kết quả đúng và tiếp tục chạy các bước tiếp theo, nếu thấy hình vẽ bộ nhớ chính như dưới đây và không thay đổi nữa, thì có thể kết luận rằng:

1. Chương trình mà chúng ta xây dựng để cộng 2 số là đúng và đã "dừng".
2. Các vi chỉ thị đã được lấy từ Control Store ra để thực hiện chương trình của chúng ta viết đã được thiết kế đúng.
3. Các thành phần của hệ thống thuộc đường dữ liệu và đơn vị điều khiển đã tham gia trực tiếp vào việc thi hành chương trình đã được xây dựng và ghép nối với nhau một cách đúng đắn.




Hình 12.4 Hình ảnh bộ nhớ chính sau khi chạy chương trình

# BÀI TẬP CÓ GỢI Ý CHO SINH VIÊN (20 BÀI)

## Bài số 1 Xây dựng bộ nhớ RAM 1K-byte từ các chip 256*8 bits

Hãy sử dụng bộ mô phỏng mạch điện số Logisim để xây dựng một bộ nhớ RAM 1K bytes bằng cách sử dụng 4 chip RAM loại 256x8 bit; Sau đó mô phỏng hoạt động (read/write) của nó. Ngoài ra, hãyviết "Hướng dẫn cho người sử dụng" đối với mạch mô phỏng một cách ngắn gọn và dễ hiểu ngay trên cửa sổ vẽ mạch điện (Canvas).

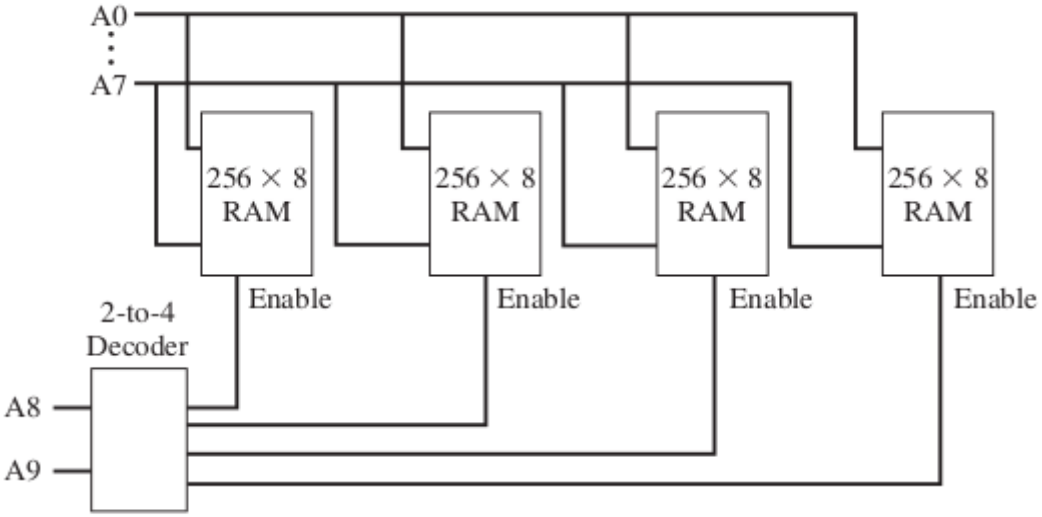



Figure 20.16 Address Decoding

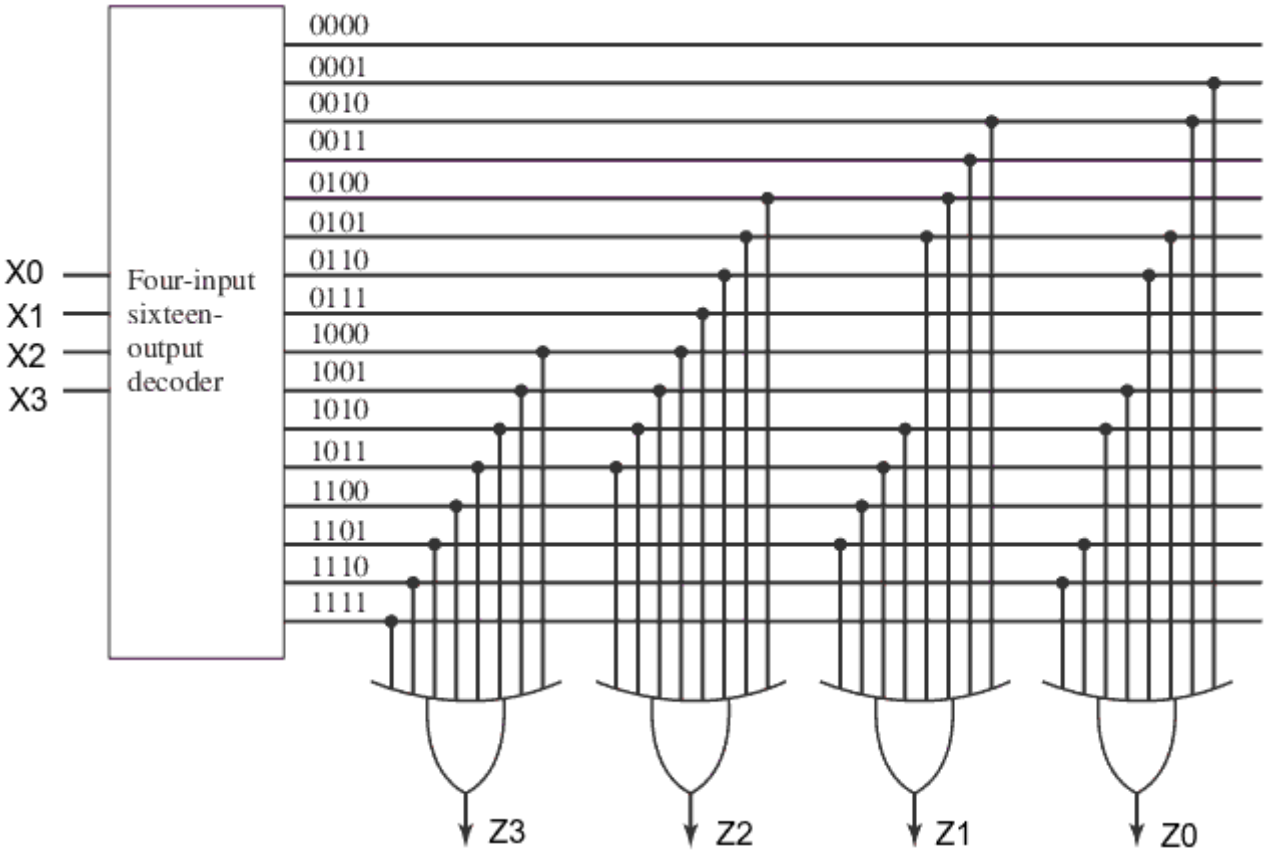
**Gợi ý:**

Xây dựng bộ nhớ theo sơ đồ ở hình vẽ trên ("Figure 20.16 Address Decoding"), sau đó bổ sung các đơn vị chức năng cần thiết, như:

- Một phần tử input 8 bits để đưa địa chỉ lên bus địa chỉ ($A_7$...$A_0$)
- Một phần tử input 2 bits để đưa vào bộ giải mã 2to4 chọn chip ($A_9A_8$)
- Một bộ giải mã 2to4 để chọn chip nhớ
- Một phần tử pin (input) 8 bits để đưa dữ liệu sẽ ghi vào RAM
- Một phần tử pin (output) 8 bits để chứa dữ liệu đọc từ RAM
- Một số phần tử pin (input) để đưa vào mạch điện các tín hiêu, như: Load, Clear, Clock, Enable…

## Bài số 2 Xây dựng một bộ nhớ ROM 16x4 bits

Hãy xây dựng một bộ nhớ ROM 16x4 bit được thể hiện trên hình vẽ sau đây và bổ sung các phần tử Hex Digit Display để hiển thị địa chỉ (4 bit $X_3X_2X_1X_0$) và kết quả đưa ra (4 bit $Z_3Z_2Z_1Z_0$); Sau đó mô phỏng hoạt động (read) của nó. Ngoài ra, hãy viết "Hướng dẫn cho người sử dụng" đối với mạch mô phỏng một cách ngắn gọn và dễ hiểu ngay trên cửa sổ vẽ mạch điện (Canvas). Chú ý: giả thiết $X_0$ là bít bậc 0 của địa chỉ 4 bit, $Z_0$ là là bít bậc 0 của đầu ra 4 bit.

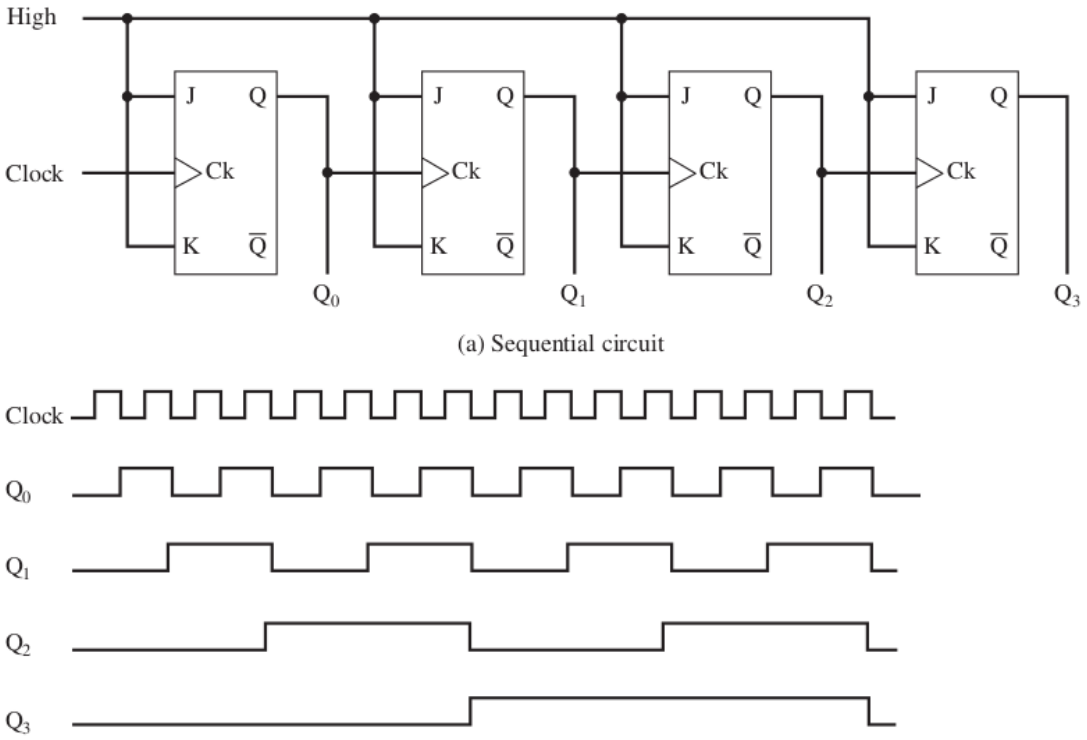



**Gợi ý:**

- Trước hết xây dựng bộ nhớ theo đúng sơ đồ ở hình vẽ.
- Sau đó bổ sung 2 phần tử Hex Digit Display để hiển thị dưới dạng số Hexa giá trị địa chỉ và giá trị đọc được từ bộ nhớ ROM. Tất nhiên, để nối phần tử Hex Digit Display với các phần tử pin (input và output) 1 bit thì cần phải sử dụng phần tử spliter có tham số "Fan Out" = 4 và Bit Width In cũng bằng 4.
- Để mô phỏng hoạt động của bộ nhớ ROM này trước hết ta cần chọn "Simulation Enable" ở thực đơn "Simulate", sau đó ta cần cho giá trị địa chỉ 4 bit lần lượt nhận từng giá trị từ nhỏ nhất (0000b) đến lớn nhất (1111b) và quan sát các đầu ra của decoder cũng như đầu ra Ouput có đúng như mạch điện không. Trong khi quan sát cần ghi các giá trị Address và Output vào một bảng.

## Bài số 3: Xây dựng bộ đếm kiểu gợn sóng (ripple counter) 8 bit

Hãy sử dụng bộ mô phỏng mạch điện số Logisim để xây dựng một bộ đếm kiểu gợn sóng 8 bit (8-bit ripple counter) được mô tả bởi hình vẽ dưới đây.




(a) Sequential circuit




(b) Timing diagram

**Figure 20.30** Ripple Counter

**Gợi ý:**

- Về bộ đếm kiểu gợn sóng 8 bit (8-bit ripple counter) như trên, có thể tham khảo cuốn sách: "Computer Organization and Architecture: Designing for Performance, Eight Edition", William Stallings, 2010; Hình trên chính là Figure 20.30.
- Bốn phần tử chính trên hình là 4 J-K Flip-Flop, có trong thư viện của Logisim. Các đầu vào điều khiển J và K nối với nhau và nối với 1 phần tử pin (input) được đặt giá trị 1 (High) để làm cho các J-K Flip-Flop có thể đếm (Enable Counting).
- Cần bổ sung các đơn vị chức năng cần thiết khác nữa vào mạch điện: (1) 1 phần tử "Clock"; (2) 1 phần tử pin (input) để có thể reset bộ đếm khi chúng ta muốn; (3) 1 phần tử Hex Digit Display để hiển thị số đưa ra dưới dạng Hexa từ 0..F; (4) 1 phần tử splitter có tham số "Fan Out" = 4 và Bit Width In cũng bằng 4 để nối phần tử Hex Digit Display với 4 bit đầu ra của bộ đếm.
- Để mô phỏng hoạt động của bộ đếm này trước hết ta cần chọn "Simulation Enable" ở thực đơn "Simulate", sau đó đặt phần tử pin nối với các đầu J, K giá trị 1 - High (Enable Counting), sau đó đặt "Ticks Enable" cũng ở thực đơn "Simulate".
- Để dừng việc mô phỏng, hãy bỏ lựa chọn "Ticks Enable" ở thực đơn "Simulate".

## Bài số 4 Xây dựng mạch điện điều khiển phần tử 7-segment display

Hãy xây dựng một mạch tổ hợp sử dụng để điều khiển phần tử 7-segment display hiển thị các số thập phân có giá trị từ 0-9 theo sơ đồ khối ở hình vẽ dưới đây. Mạch điện có 4 lối vào, chúng tạo ra mã 4 bit được sử dụng trong cách biểu diễn số thập phân kiểu nén - mỗi byte biểu diễn 1 số thập phân có 2 chữ số, bằng cách sử dụng 2 nhóm 4 bit để biểu diễn 2 chữa số thập phân ($0_{10}$ = 0000, ..., $9_{10}$ = 1001). Bảy lối ra xác định đoạn nào sẽ được kích hoạt để hiển thị một số thập phân đã cho. Chú ý rằng có một số tổ hợp các lối vào và lối ra không được sử dụng.

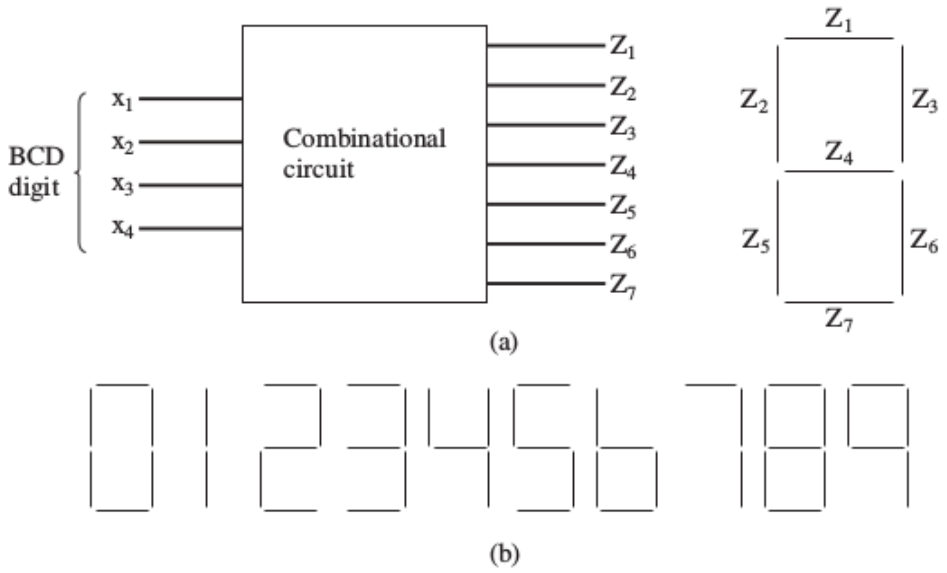



Figure 20.36 Seven-Segment LED Display Example

**Gợi ý:**

- Về mạch điện điều khiển phần tử 7-segment display có thể tham khảo cuốn sách: "Computer Organization and Architecture: Designing for Performance, Eight Edition", William Stallings, 2010; Hình trên chính là Figure 20.36.
- Đơn vị có ghi nhãn "Combimational circuit" ở góc trên bên trái hình a là đơn vị chức năng mà chúng ta cần xây dựng, nó có 7 đầu ra $Z_1..Z_7$ sẽ nối với 7 đầu vào của phần tử hiển thị 7-Segment Display, được mô tả ở góc trên bên phải của hình a.
- Để xây dựng đơn vị chức năng "Combimational circuit", trước hết chúng ta cần lập bảng chân lý, với các cột $X_1$, $X_2$, $X_3$, $X_4$ và các cột $Z_1..Z_7$, trong đó $Z_i$ là hàm logic của 4 biến đầu vào $X_1$, $X_2$, $X_3$, $X_4$. Sau đó chúng ta cũng có thể tối thiểu hóa hàm logic, rồi xây dựng mạch logic dựa trên bảng chân lý.
- Để mô phỏng hoạt động của mạch điện điều khiển 7-Segment Display trước hết ta cần chọn "Simulation Enable" ở thực đơn "Simulate", sau đó ta cần cho giá trị đầu vào 4 bit lần lượt nhận từng giá trị từ nhỏ nhất (0000b) đến lớn nhất (1001b) và quan sát số được hiển thị bởi phần tử 7-Segment Display xem có đúng không.

## Bài số 5 Xây dựng bộ chuyển mã nhị phân 3 bit sang mã Gray 3 bit

Hãy xây dựng mạch điện thực hiện chức năng chuyển mã nhị phân 3 bit thành mã Gray 3 bit. Mã Gray còn có một vài tên gọi khác là: "reflected" code, hay "binary reflected Gray code". Mã Gray là một mã nhị phân cho số nguyên. Nó khác với cách biểu diễn số nhị phân thông thường ở chỗ, chỉ có một bit thay đổi giữa 2 biểu diễn của 2 số liên tiếp bất kỳ. Điều này là có ích cho các ứng dụng chẳng hạn như các bộ đếm hoặc các bộ chuyển đổi tương tự – số, trong đó có một dãy các con số được sinh ra. Bởi vì mỗi lần (sinh ra một con số) chỉ có một bit thay đổi, nên không bao giờ có chuyện nhập nhằng (nhầm) do sự sai khác về thời gian là ít.

Để chuyển đổi một số nhị phân $b_1b_2 \ldots b_{n-1}b_n$ thành mã Gray, hãy bắt đầu từ bit bên phải $b_n$ (bit cuối). Nếu $b_{n-1}=1$ thì thay $b_n = 1-b_n$, ngược lại (tức là $b_{n-1}=0$) thì không thay đổi. Tiếp tục thực hiện như trên với $b_{n-1}$ cho đến chữ số đầu tiên là $b_1$, với $b_1$ nó được giữ không thay đổi vì $b_0$ được giả thiết là bằng 0. Dãy số (nhị phân) sinh ra $g_1g_2 \ldots g_{n-1}g_n$ chính là mã Gray tương ứng (reflected binary Gray code).

Bảng dưới đây là thí dụ chuyển đổi số nhị phân 3 bit thành mã Gray 3 bit.

| **Binary Code ($b_1b_2b_3$)** | **Gray Code ($d_1d_2d_3$)** |
|---|---|
| 000 | 000 |
| 001 | 001 |
| 010 | 011 |
| 011 | 010 |
| 100 | 110 |
| 101 | 111 |
| 110 | 101 |
| 111 | 100 |

**Gợi ý:**

- Đây chính là bài tập 20.11 trong cuốn sách: "Computer Organization and Architecture: Designing for Performance, Eight Edition", William Stallings, 2010.
- Cách thứ nhất để xây dựng mạch chuyển mã: Việc chuyển mã theo yêu cầu luôn cần đến kết quả của việc so sánh giữa bit bậc 3 và bit bậc 2 và giữa bit bậc 2 và bit bậc 1 của số nhị phân 3 bit đưa vào, vì vậy chúng ta có thể sử dụng 2 cổng XOR. Riêng với bit bậc 1 (thấp nhất) của số nhị phân 3 bit đưa vào, chỉ cần đưa thẳng ra làm bit bậc 1 của mã Gray 3 bit.
- Cách thứ hai để xây dựng mạch chuyển mã: Chúng ta cũng có thể lập bảng chân lý như ở bảng trên, với các cột $b_1$, $b_2$, $b_3$ và các cột $g_1$, $g_2$ và g3, trong đó $g_i$ là hàm logic của 3 biến đầu vào $d_1$, $d_2$, $d_3$. Sau đó chúng ta cũng có thể tối thiểu hóa hàm logic, rồi xây dựng mạch logic dựa trên bảng chân lý.
- Để mô phỏng hoạt động của mạch chuyển mã nhị phân 3 bit thành mã Gray 3 bit trước hết ta cần chọn "Simulation Enable" ở thực đơn "Simulate", sau đó ta cần cho giá trị đầu vào 3 bit lần lượt nhận từng giá trị từ nhỏ nhất (000b) đến lớn nhất (111b) và quan sát mã Gray 3 bit ở đầu ra xem có đúng như ở bảng trên không.

## Bài số 6 Xây dựng bộ chuyển mã Gray 3 bit sang mã nhị phân 3 bit

Hãy xây dựng mạch điện thực hiện chức năng chuyển mã Gray 3 bit thành mã nhị phân 3 bit. Mã Gray còn có một vài tên gọi khác là: "reflected" code, hay "binary reflected Gray code". Mã Gray là một mã nhị phân cho số nguyên. Nó khác với cách biểu diễn số nhị phân thông thường ở chỗ, chỉ có một bit thay đổi giữa 2 biểu diễn của 2 số liên tiếp bất kỳ. Điều này là có ích cho các ứng dụng chẳng hạn như các bộ đếm hoặc các bộ chuyển đổi tương tự – số, trong đó có một dãy các con số được sinh ra. Bởi vì mỗi lần (sinh ra một con số) chỉ có một bit thay đổi, nên không bao giờ có chuyện nhập nhằng (nhầm) do sự sai khác về thời gian là ít.

Bảng dưới đây là thí dụ chuyển đổi số nhị phân 3 bit thành mã Gray 3 bit.

| **Binary Code ($b_1b_2b_3$)** | **Gray Code ($d_1d_2d_3$)** |
|---|---|
| 000 | 000 |
| 001 | 001 |
| 010 | 011 |
| 011 | 010 |
| 100 | 110 |
| 101 | 111 |
| 110 | 101 |
| 111 | 100 |

Để chuyển đổi một mã Gray $g_1g_2 \dots g_{n-1}g_n$ thành mã nhị phân, hãy bắt đầu từ bit bậc n và tính:

$$\Sigma_n \equiv \sum_{i=1}^{n-1} g_i \pmod 2.$$

Nếu $\Sigma_n = 1$, thì $b_n = 1-g_n$; Ngược lại thì không thay đổi ($b_n := g_n$).

Tiếp theo tính:

$$\Sigma_{n-1} \equiv \sum_{i=1}^{n-2} g_i \pmod 2,$$

và cứ lặp lại tương tự như vậy. Con số nhận được $b_1b_2 \dots b_{n-1}b_n$ chính là số nhị phân ứng với mã Gray đưa vào.

**Gợi ý:**

- Theo phương pháp chuyển mã Gray thành mã Binary trình bày ở trên, ta có thể nhận xét: bit bi là kết quả phép toán XOR của g1...gi. Như vậy bi có thể nhận được ở đầu ra của cổng XOR có i đầu vào nối với g1...gi. Còn b1 = g1, vì với giả thiết g0=1 thì b1 = g1 XOR 0 = g1.
- Để mô phỏng hoạt động của mạch chuyển mã Gray 3 bit thành mã Binary 3 bit trước hết ta cần chọn "Simulation Enable" ở thực đơn "Simulate", sau đó ta cần cho giá trị đầu vào 3 bit lần lượt nhận từng giá trị từ nhỏ nhất (000b) đến lớn nhất (111b) và quan sát mã Binary 3 bit ở đầu ra xem có đúng như ở bảng trên không.

## Bài số 7 Xây dựng một full adder sử dụng 5 cổng

Hãy xây dựng một full adder sử dụng 5 cổng có chức năng tương đương một full adder cho trước ở hình vẽ dưới đây.

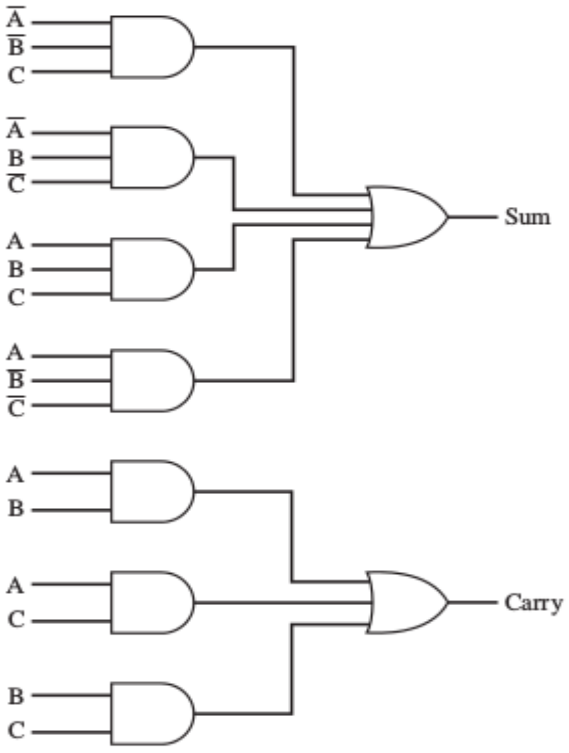



Figure 20.20 **Implementation of an Adder**

**Gợi ý:**

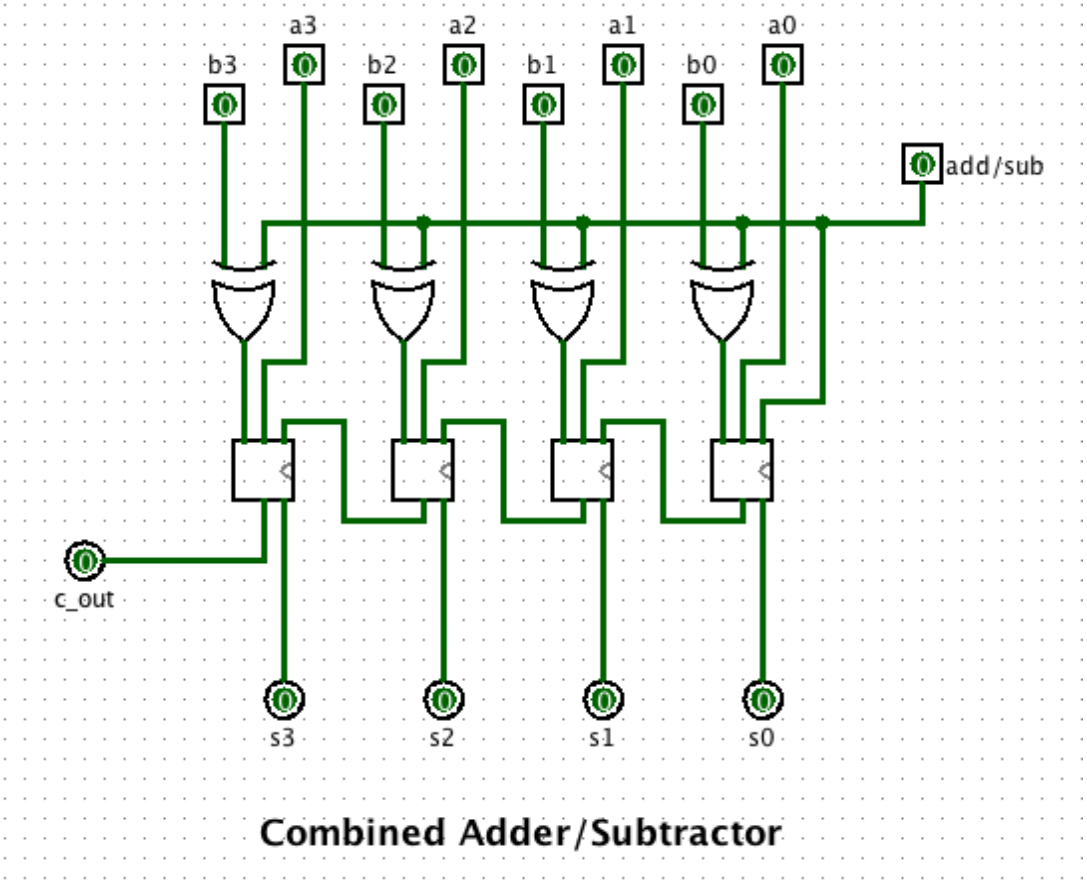
- Tại mục 3.2.3.2 Bộ cộng - Adder của GT KTMT đã có hình vẽ và phân tích về full adder được xây dựng từ 5 cổng (2 cổng XOR, 2 cổng AND và 1 cổng OR), với 3 toán hạng 1-bit đưa vào là a, b và Carry-In. Bộ full adder đó có chức năng hoàn toàn giống bộ full adder ở hình vẽ trên, vì vậy chúng ta chỉ cần xây dựng full adder ở hình 3-12 tại mục 3.2.3.2 nói trên.
- Để thuận tiện cho việc mô phỏng, chúng ta sử dụng phần tử pin (input) nối với các đầu vào A, B và Carry-In, sử dụng phần tử pin (output) nối với các đầu ra Sum và Carry-Out.
- Để mô phỏng hoạt động của bộ full adder này trước hết ta cần chọn "Simulation Enable" ở thực đơn "Simulate", sau đó ta cho giá trị 3 bit vào (A, B và Carry-In) lần lượt nhận từng giá trị từ nhỏ nhất (000b) đến lớn nhất (111b) và quan sát các đầu ra Sum và Carry-Out xem có đúng không.

## Bài số 8 Xây dựng ALU 4-bit với các chức năng: add, sub, dec và inc

Hãy xây dựng một ALU có thể thực hiện các phép toán add (cộng), sub (trừ) dec (giảm 1) và inc (tăng 1) trên các toán hạng 4 bit. ALU này được xây dựng từ các cổng logic cơ bản và phần tử Adder có sẵn trong thư viện của Logisim. Ngoài ra hãy sử dụng các phần tử pin (input/output), splitter (bó dây) và phần tử Hex Digit Display để hiển thị các toán hạng đưa vào và kết quả nhận được.

**Gợi ý:**

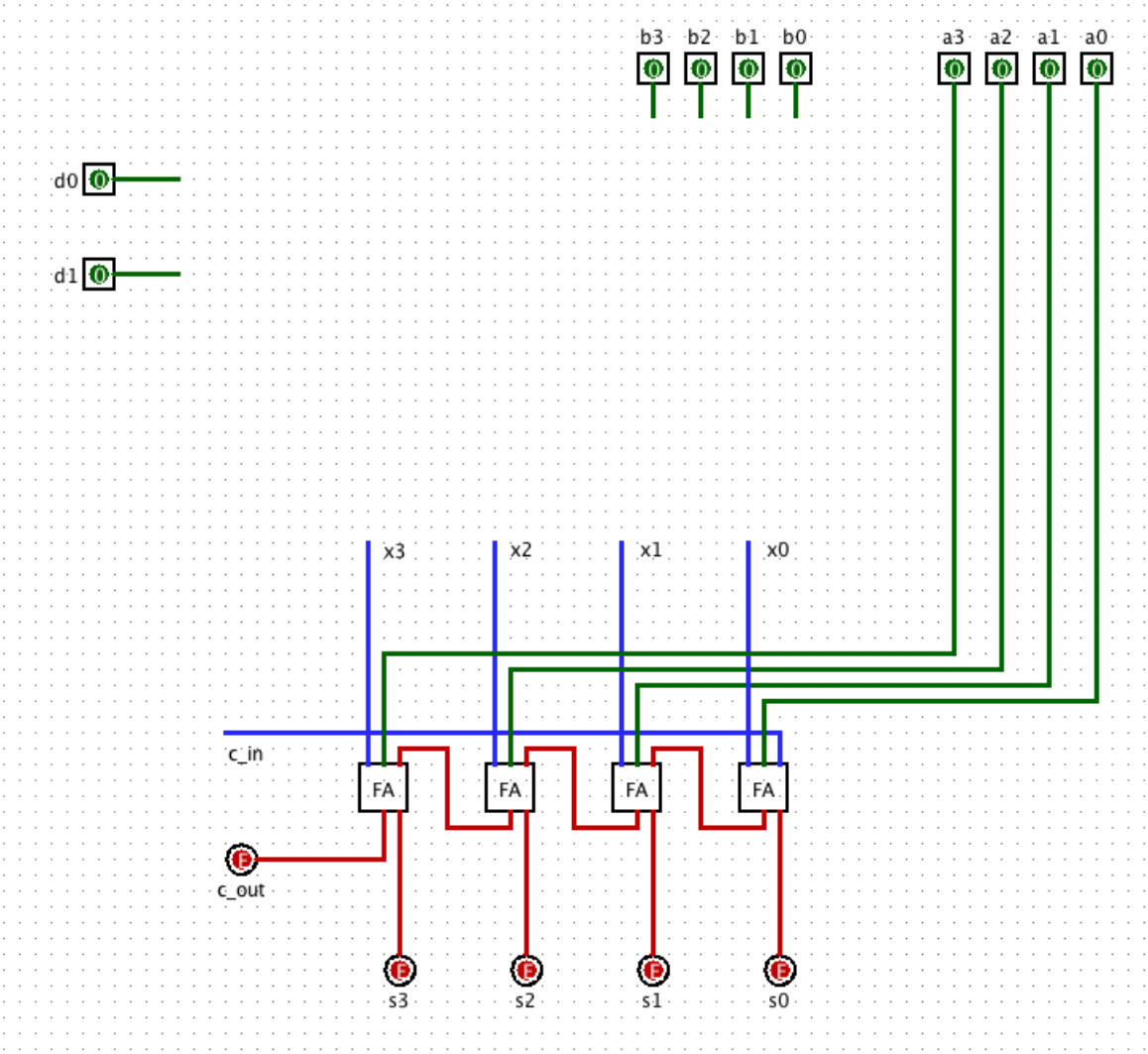
- ALU 4-bit có thể được xây dựng dựa trên thành phần chính là một đơn vị kết hợp Adder/Subtractor theo sơ đồ sau đây. Nó bao gồm 4 bộ adder 1 bit (tạo thành 4-bit ripple-carry adder) và các cổng XOR, các phần tử pin (input và output). Phần tử pin (input) có ghi nhãn "add/sub" dùng để xác định chức năng cộng (add) hay trừ (sub).




- Để thực hiện được 4 phép toán, ALU 4-bit của chúng ta phải có 2 đường điều khiển, giả sử chúng ta ký hiệu là $d_1$ và $d_0$, với các giá trị 00, 01, 10, 11 thì ALU sẽ thực hiện các phép toán tương ứng là add (a+b), sub(a-b), dec (a-1) và inc (a+1). Ta thấy $a_3$, $a_2$, $a_1$, $a_0$ luôn được đưa vào 4 bộ 1-bit adder (tạo nên 4-bit adder/subtracter), như vậy nhiệm vụ còn lại của chúng ta thực chất là xác định xem sẽ nối như thế nào 4 đầu vào 1 bit còn lại của adder/subtracter (ở hình dưới đây ký hiệu là $x_3$, $x_2$, $x_1$, $x_0$) với các đường điều khiển $d_1$, $d_0$ và 4 bit của toán hạng vào thứ 2 ($b_3$ $b_2$ $b_1$ $b_0$). Để thực hiện việc nối dây nêu trên, chúng ta phân tích tiếp như sau:
  - $d_1d_0 = 00$ (ADD). Khi đó chúng ta cần phải nối $b_i$ với $x_i$.

– $d_1d_0$ = 01 (SUB). Khi đó chúng ta cần phải nối $b_i$ đảo với $x_i$ đồng thời thiết lập carry-in (“c_in”) bằng 1. Thực chất là truyền vào x ($x_3$ $x_2$ $x_1$ $x_0$) giá trị nhị phân âm dạng bù 2 của b ($b_3$ $b_2$ $b_1$ $b_0$).

– $d_1d_0$ = 10 (DEC). Với phép toán này chúng ta không cần giá trị b ($b_3$ $b_2$ $b_1$ $b_0$); Chúng ta sẽ đưa vào x ($x_3$ $x_2$ $x_1$ $x_0$) giá trị 4 bit là 1111 (thực chất là -1 dạng nhị phân bù 2) để cộng với a ($a_3$ $a_2$ $a_1$ $a_0$). Như vậy khi thực hiện phép toán DEC chúng ta cần thiết lập cho $x_i$=1 và carry-in (“c_in”) bằng 0.




– $d_1d_0$ = 11 (INC). Giống như với phép toán DEC, với INC chúng ta không cần giá trị b ($b_3$ $b_2$ $b_1$ $b_0$). Để cộng a ($a_3$ $a_2$ $a_1$ $a_0$) với 1, chúng ta chỉ cần thiết lập $x_i$=0 và carry-in (“c_in”) bằng 1.

- Từ các phân tích trên, chúng ta có thể lập bảng chân lý như dưới đây, với các giá trị "vào" (biến) là $d_1$, $d_0$ và bi, các giá trị "ra" (hàm) là $x_i$ and c_in:

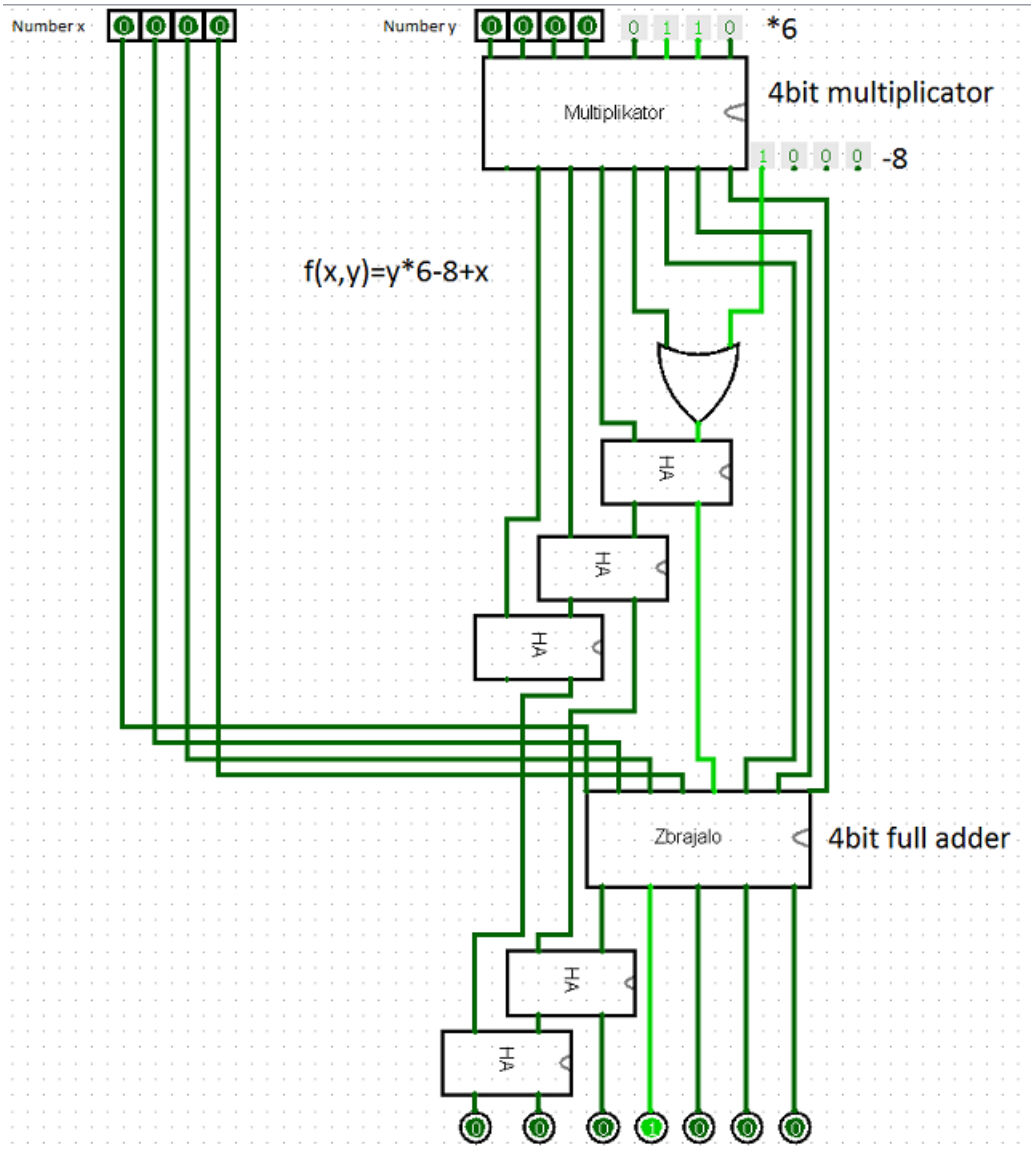
| $d_1$ | $d_0$ | $b_i$ | $x_i$ | c_in |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |

- **Các việc cần làm tiếp theo:**
  - Vận dụng các kiến thức đã học về đại số logic, dựa trên bảng chân lý vừa lập, viết biểu thức $x_i$ như hàm của $d_1$, $d_0$ và $b_i$. Sau đó đơn giản hóa biểu thức.
  - Tương tự với $x_i$, viết biểu thức c_in như hàm của $d_1$, $d_0$ và $b_i$. Sau đó đơn giản hóa biểu thức.
  - Chạy Logisim, "thi công" mạch điện dựa trên các hình vẽ gợi ý và các phân tích về $x_i$ và c_in.
  - Chạy thử ALU và kiểm tra việc thực hiện các phép toán.
  - Nhớ lưu (Save) mạch điện vào file một cách thường xuyên.

## Bài số 9 Xây dựng mạch điện tính giá trị của hàm f(x,y) = 6*y-8+x với biến và hằng là 4 bits

**Gợi ý:**

- Mạch điện thực hiện chức năng theo yêu cầu có thể được xây dựng theo sơ đồ khối như ở hình vẽ dưới đây. Sơ đồ này chủ yếu để thể hiện ý tưởng giải bài toán, khi xây dựng mạch điện, nếu chúng ta sử dụng các phần tử Multiplier, Adder có sẵn trong thư viện của Logisim, thì mạch điện có thể sẽ đơn giản hơn.




- Đơn vị chức năng có ghi nhãn "multiplikator" là bộ nhân 4 bit, thực hiện việc nhân y là số 4 bit với hằng số 6 (0110), tuy  là số 4 bit nhưng có bit cao bằng 0 nên sinh ra kết quả là số 7 bit. Nếu chúng ta muốn xây dựng mạch điện tính giá trị của hàm tổng quát hơn, cụ thể là y được nhân với hằng số 4 bit, có thể nhận giá trị với bit cao nhất (bậc 3) bằng 1, thí dụ giá trị lớn nhất là 1111b (15), thì bộ nhân phải có đầu ra 8 bit. Trên hình vẽ, bộ nhân có đầu ra 8 bit nhưng bit cao nhất được bỏ qua.

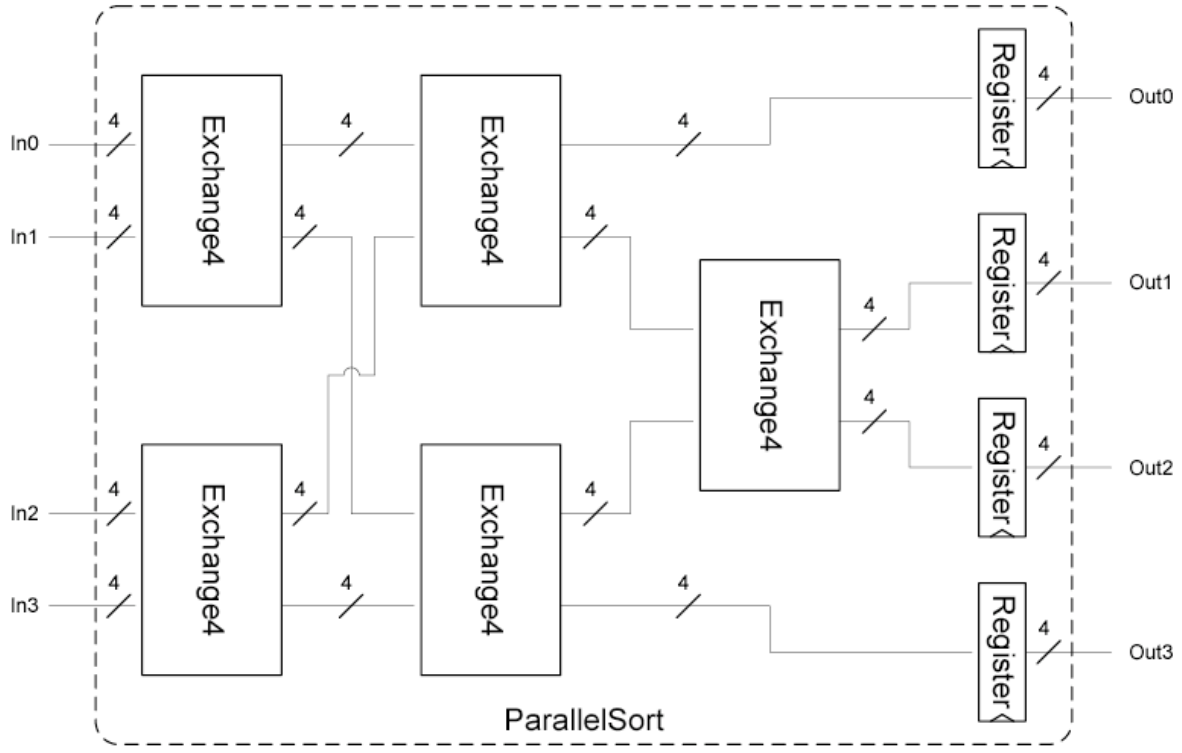
- Khối bao gồm 1 phần tử OR và 3 phần tử HA (Haft-Adder) thực hiện việc cộng kết quả của phép nhân y*6 (số 7 bit) với -8 (số 4 bit). Ở đây cần chú ý 2 điều là:
  - +8d = 1000b, -8d đổi sang dạng nhị phân âm bù 2 cũng là 1000b.
  - Kết quả của việc cộng 1 số 7 bit với 1 số 4 bit có thể sinh ra kết quả là 1 số 8 bit (bit bậc 7 bằng 1); Tuy nhiên trên hình vẽ không sử dụng bit nhớ Carry-out của việc cộng bit bậc cao nhất. Vì vậy, sinh viên hãy chú ý và xử lý vấn đề này.
- Đơn vị chức năng có ghi nhãn "Zbrajalo" là bộ cộng (full adder) 4 bit, thực hiện việc cộng toán hạng x 4 bit với 4 bit thấp nhất của kết quả tính y*6-8. Số nhớ sinh ra được được cộng với bit bậc cao hơn (bậc 4) của nhóm các bit cao của kết quả tính y*6-8, việc cộng được thực hiện nhờ các đơn vị HA (Haft Adder). Trên hình vẽ, nhóm bit cao này gồm 2 bit, như vậy là không tổng quát, như đã được nhận xét ở trên.
- **Các việc cần làm tiếp theo:**
  - Chạy Logisim, "thi công" mạch điện dựa trên sơ đồ khối và các phân tích ở trên.
  - Đưa vào các số liệu khác nhau của x và y, quan sát kết quả tính hàm f(x,y) = 6*y-8+x.
  - Nhớ lưu (Save) mạch điện vào file một cách thường xuyên.

## Bài số 10 Xây dựng mạch điện ParallelSort

Hãy xây dựng mạch điện thực hiện chức năng sắp xếp song song (ParallelSort) theo chiều giảm của 4 số 4 bit đưa vào. Sắp xếp song song được hiểu là việc sắp xếp được hoàn thành trong 1 chu kỳ đồng hồ.

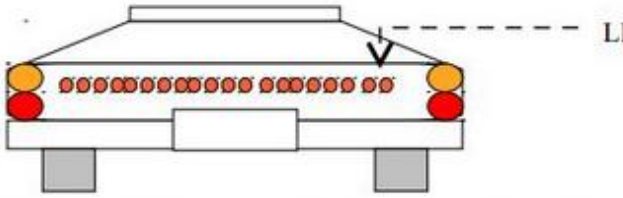
**Gợi ý:**

- Mạch điện thực hiện chức năng theo yêu cầu có thể được xây dựng theo sơ đồ khối như ở hình vẽ dưới đây.
- Đơn vị "Exchange4" thực hiện sắp xếp 2 số 4 bit đầu vào. Như vậy đơn vị này cần so sánh 2 số đầu vào và dựa trên kết quả của việc so sánh sẽ chuyển mỗi đầu vào ra đầu ra thích hợp. Trong thư viện của Logisim có phần tử Comparator có chức năng so sánh.
- Đơn vị "Register" chứa kết quả ra, có thể sử dụng phần tử pin (Output).
- Nên nối mỗi đầu vào và đầu ra với một phần tử Hex Digit Display để tiện cho việc đọc giá trị các con số nhị phân.

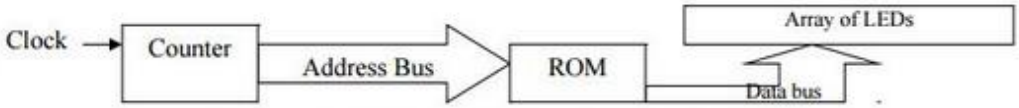


## Bài số 11: Xây dựng hệ thống đèn xi-nhan phía đuôi ô tô với LED được điều khiển bằng ROM

Hãy xây dựng hệ thống đèn xi-nhan phía đuôi ô tô theo mô tả ở hình vẽ dưới đây. Hệ thống gồm một dãy 20 đèn LED lần lượt sáng lên từng đèn, từ trái qua phải nếu nút bấm "LR" được ấn, từ phải qua trái nếu nút "RL" được ấn. Có 2 đèn vàng, đèn vàng bên trái sẽ nhấp nháy liên tục khi ấn nút "RL", đèn vàng bên phải sẽ nhấp nháy liên tục khi ấn nút "LR". Hai đèn màu đỏ trên hình là đèn phanh, khi ấn nút "BR" sẽ sáng cả 2, nhả "BR" thì cùng tắt. Các đèn (trừ đèn phanh" chỉ sáng trong 0,5s, các đèn sáng nhấp nháy sẽ luôn phiên sáng, tối sau mỗi khoảng 0,5s.




**LEDs at the back of the car for the running light effect**




**Alternative system architecture using a ROM**

*Activities to do-*

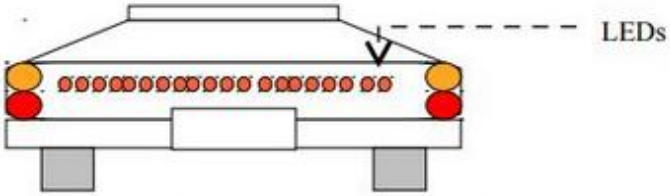
1. Draw the finite state machine representation of the required sequential circuit.
2. Use flip-flops and logic gates to implement the required circuit. The design should have a clearly identifiable data bus.

**Gợi ý:**

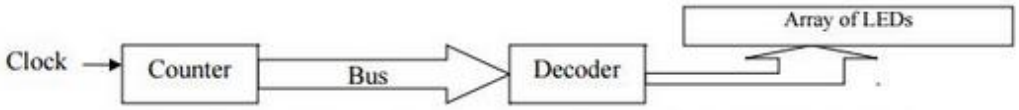
- Trên hình đã có một số gợi ý bằng tiếng Anh:
  - "Alternative system architecture using a ROM"
  - "Activities to do-"
- Chip ROM có address bus 5 dây (5 bit), kích thước word 20 bit, mỗi bit mô tả trạng thái sáng/tối của một đèn LED, như vậy trong 1 word chỉ có 1 bit 1 còn lại là 19 bit 0. Địa chỉ đưa vào chip ROM sẽ được thay đổi kiểu tuần hoàn theo "clock" của Logisim, từ 00000 đến 10011.

## Bài số 12: Xây dựng hệ thống đèn xi-nhan phía đuôi ô tô với LED được điều khiển bằng decoder

Hãy xây dựng hệ thống đèn xi-nhan phía đuôi ô tô theo mô tả ở hình vẽ dưới đây. Hệ thống gồm một dãy 20 đèn LED lần lượt sáng lên từng đèn, từ trái qua phải nếu nút bấm "LR" được ấn, từ phải qua trái nếu nút "RL" được ấn. Có 2 đèn vàng, đèn vàng bên trái sẽ nhấp nháy liên tục khi ấn nút "RL", đèn vàng bên phải sẽ nhấp nháy liên tục khi ấn nút "LR". Hai đèn màu đỏ trên hình là đèn phanh, khi ấn nút "BR" sẽ sáng cả 2, nhả "BR" thì cùng tắt. Các đèn (trừ đèn phanh" chỉ sáng trong 0,5s, các đèn sáng nhấp nháy sẽ luôn phiên sáng, tối sau mỗi khoảng 0,5s.




**LEDs at the back of the car for the running light effect**




**System architecture of the required circuit designed with a decoder**

*Activities to do-*

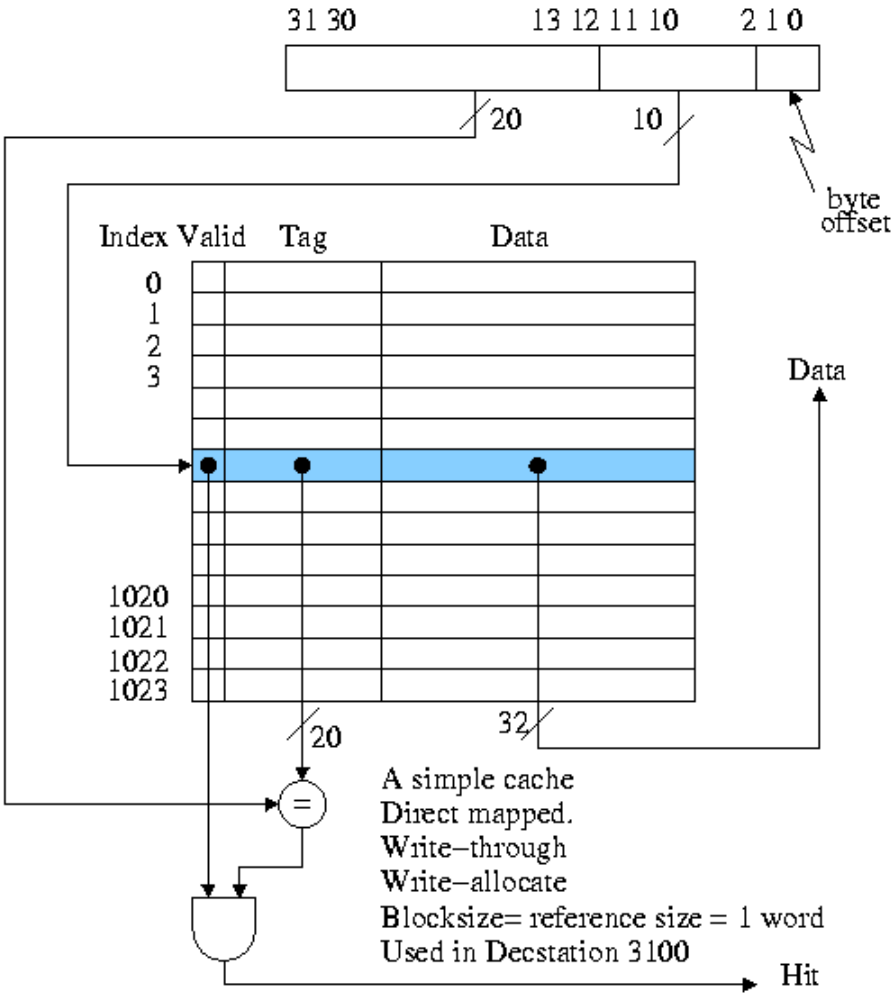
1. Draw the finite state machine representation of the required sequential circuit.
2. Use flip-flops and logic gates to implement the required circuit. The design should have a clearly identifiable data bus.

**Gợi ý:**

- Trên hình đã có một số gợi ý bằng tiếng Anh:
  - "System architecture of the required circuit designed with a decoder"
  - "Activities to do-"
- Decoder có đường tín hiệu vào 5 dây (5 bit), như vậy sẽ có 2^5 = 32 đầu ra, tuy nhiên để điều khiển 20 đèn LED chúng ta chỉ sử dụng 20 đầu ra, từ 0 đến 19, các đầu ra còn lại không được sử dụng. Địa chỉ đưa vào chip đầu select của decoder sẽ được thay đổi kiểu tuần hoàn theo "clock" của Logisim, từ 00000 đến 10011. Tất nhiên nếu đầu vào thay đổi tử 0000 đến 11111 thì nói chung cũng không ảnh hưởng đến chức năng mà chúng ta cần.

## Bài số 13: Xây dựng mạch logic số mô phỏng hoạt động của bộ nhớ Direct mapped cache

Cho bộ nhớ cache được mô tả bằng hình vẽ dưới đây. Hãy bổ sung các phần tử logic để có thể đọc hoặc ghi bộ nhớ cache. Khi ghi vào cache: trích 10 bit "cache slot" từ địa chỉ 32 bit, sau đó ghi dữ liệu 32 bit vào trường Data, ghi giá trị tag vào trường Tag và ghi giá trị 1 vào trường Valid. Khi đọc bộ nhớ cache: trích 10 bit từ bit 2-11 để chọn slot (đánh địa chỉ slot), nếu bit Valid bằng 1 và kết quả so sánh trường tag của địa chỉ với nội dung trường tag trong cache entry tương ứng là 1 (true), thì sinh tín hiệu "Cache Hit" bằng 1 và đưa 32 bit dữ liệu ra. Ngoài ra bổ sung các phần tử hiển thị số Hexa để đọc đọc dữ liệu 32 bit đưa ra (đầu ghi "Data").




**Gợi ý:**

Trong thư viện Logisim không có phần tử nào có thể dùng làm cache hoặc cache slot (kích thước có thể thay đổi tùy ý và chia thành 3 trường cùng địa chỉ nhưng có thể ghi độc lập); Vì vậy, hãy sử dụng 3 chip RAM làm bộ nhớ cache: 1Kx1 bits, 1Kx20 bits và 1Kx32 bits và nối các đầu vào địa chỉ của 3 chip này với nhau.

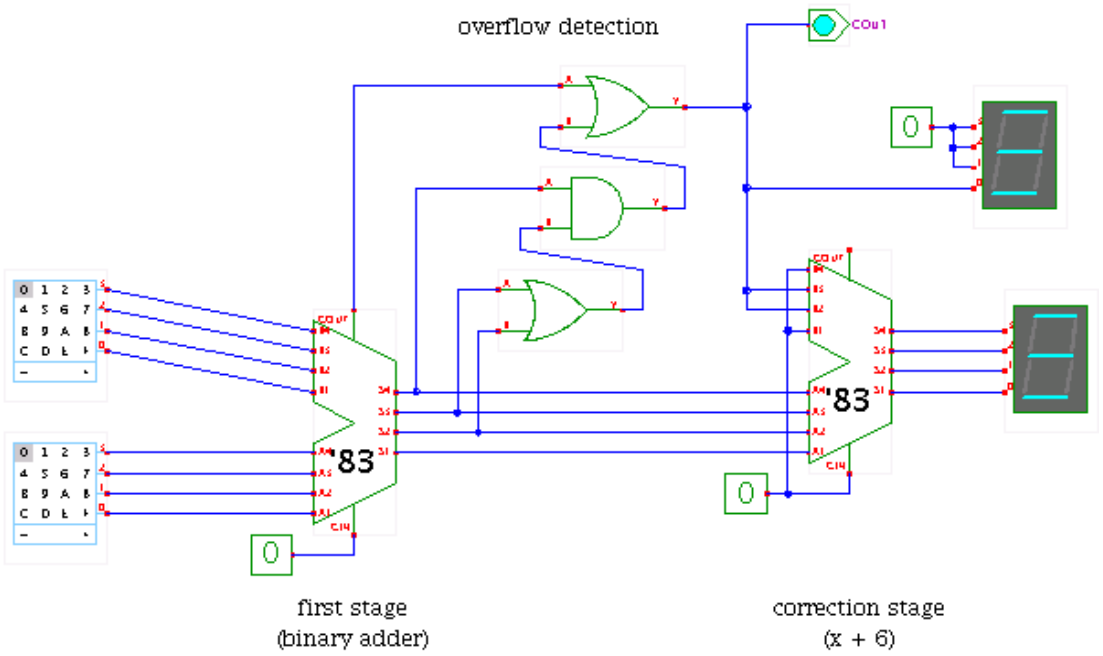
Khi mô phỏng để kiểm tra tính đúng đắn của mạch điện, hãy tiến hành ghi (write) trước vào một vài slots, sau đó đọc lại các slot đó và một số slot lân cận.

## Bài số 14: Xây dựng bộ cộng BCD 4 bit

Hãy xây dựng bộ cộng 2 số BCD (Binary Coded Decimal) 4 bit, có hiển thị các số đưa vào và kết quả dưới dạng số BCD.

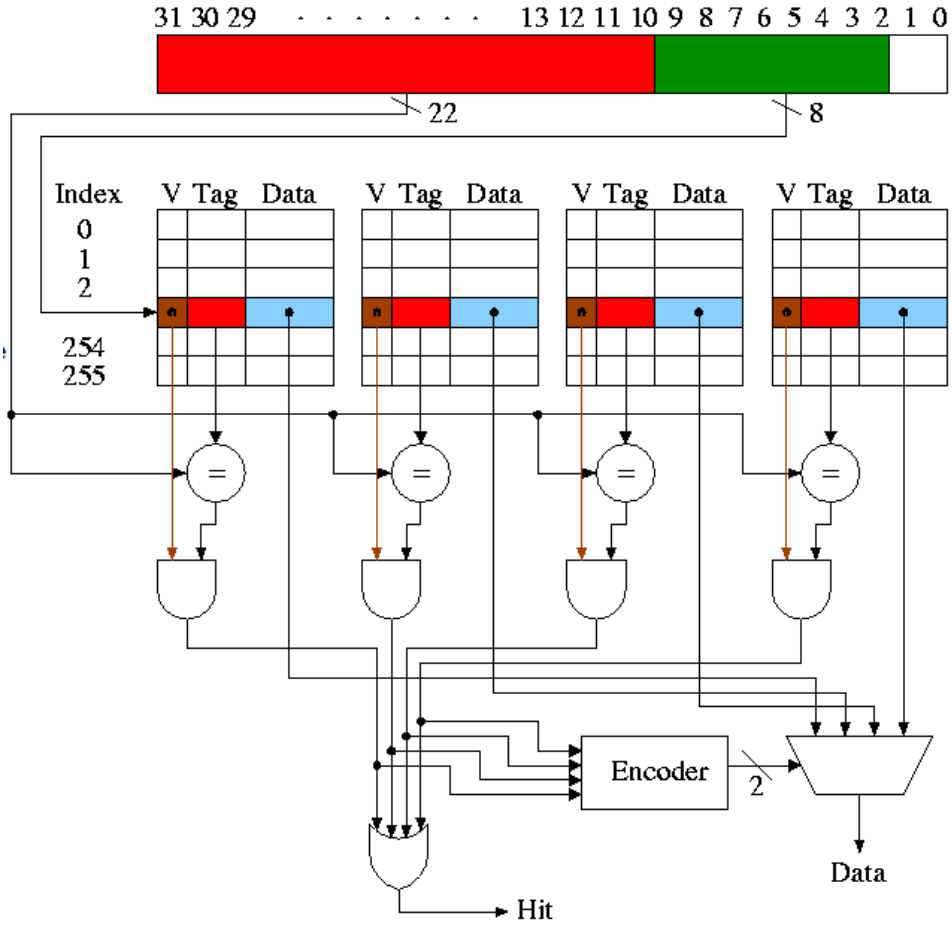
**Gợi ý:**

- Số BCD (Binary Coded Decimal) là một loại mã trong đó 1 byte (8 bit) biểu diễn 2 số thập phân, mỗi nhóm 4 bit biểu diễn 1 số thập phân có giá trị từ 0-9. Vì 1 số nhị phân 4 bit tương ứng với các giá trị thập phân từ 0-15 (hay 0-F dạng hexa) nên chỉ có 10 giá trị (nhị phân) từ 0000 đến 1001 là hợp lệ, 6 giá trị lớn hơn, từ 1001 đến 1111 là không hợp lệ.
- Có thể xây dựng mạch điện theo yêu cầu dựa trên sơ đồ sau. Đây là mạch điện thường được sử dụng trong các bộ vi xử lý có các chỉ thị tính toán số BCD. Phép cộng số BCD gồm 2 bước. Bước 1 là phép cộng nhị phân thông thường 2 số 4 bit, kết quả sẽ là một số 5 bit có giá trị lớn nhất là 10010 khi cộng 2 số BCD 4 bit lớn nhất là 1001 (9). Bước 2 khi kết quả của phép cộng ở bước 1 (số 4 bit) lớn hơn 9, thì đem kết quả này cộng với 6 (0110), thực chất là để sinh ra số BCD thứ nhất (4 bit thấp của kết quả là số 5 bit) hợp lệ (không lớn hơn 9).
- Ở bước 2 đồng thời phải xác định (tính) xem bit cao nhất (bit bậc 4) có bằng 1 hay không. Qua các phân tích ở trên có thể thấy, bit cao nhất bằng một trong hai trường hợp, liên quan đến số nhớ (C_out =1) sinh ở ở bước 1:
  - C_out=1
  - C_out=0, nhưng kết quả lớn hơn 9 (1001)
- Phần tử Hex Digit Display có input 4 bit, vì vậy để đưa bit 1 của số BCD thứ 2 vào ta cần dùng 1 phần tử spliter, dây 0 của nó nối với bit cao nhất nói trên, còn các dây 1, 2, 3 nối với phần tử Constant được khởi tạo giá trị 0.



## Bài số 15: Xây dựng mạch logic số mô phỏng hoạt động đọc của bộ nhớ Set-associative cache

Cho bộ nhớ cache kiểu Set-associative được mô tả bằng hình vẽ dưới đây. Hãy bổ sung các phần tử logic để có thể nạp địa chỉ vào thanh ghi 32 bit; so sánh trường tag của địa chỉ với nội dung các (4) trường tag trong các (4) cache entry tương ứng; truyền dữ liệu tìm thấy trong cache (chứa trong trường "Data") ra ngoài. Ngoài ra bổ sung các phần tử hiển thị số Hexa để đọc địa chỉ và đọc dữ liệu 32 bit đưa ra (đầu ra của bộ demultiplexer có ghi nhãn "Data").

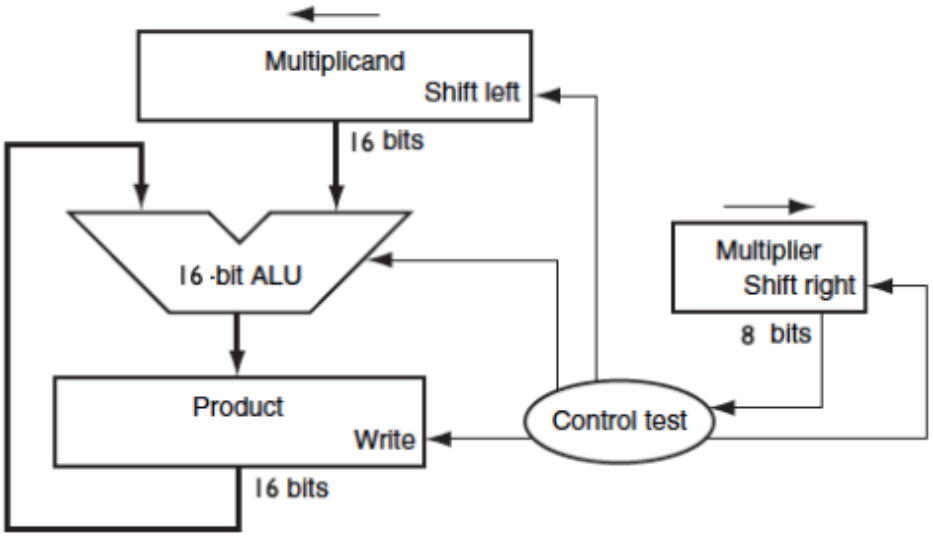



**Gợi ý:**

- Có thể xây dựng mạch điện theo sơ đồi khối như trên. Để mạch điện không quá phức tạp, bài tập này chỉ yêu cầu mô phỏng hoạt động đọc bộ nhớ cache.
- Trong thư viện Logisim không có phần tử nào có thể dùng làm cache hoặc cache slot (kích thước có thể thay đổi tùy ý và chia thành 3 trường cùng địa chỉ nhưng có thể ghi độc lập). Vì vậy, hãy xây dựng một mạch con (subcircuit) sử dụng 3 chip RAM làm bộ nhớ cache: 256x1 bits chứa các bit Valid, 256x22 bits chứa các giá trị Tag number và 256x32 bits chứa data và nối các đầu vào địa chỉ của 3 chip này với nhau.
- Khi mô phỏng hoạt động đọc (read) theo yêu cầu, hãy tiến hành nạp nội dung (write) trước vào một vài slots (kích nút chuột phải vào chip RAM, rồi chọn "Edit Content"), sau đó đọc lại các slot đó và một số slot lân cận.

## Bài số 16: Xây dựng mạch điện nhân 2 số 8 bit

Hãy xây dựng mạch điện thực hiện chức năng nhân 2 số 8 bit theo sơ đồ khối dưới đây.




**Yêu cầu:**

1. Không được sử dụng phần tử Multiplier có sẵn trong thư viện của Logisim;
2. Mạch điện bao gồm Main circuit và một hoặc một vài Subcircuit. Main circuit bao gồm các phần tử pin (input và output) chứa số bị nhân (Multiplicand), số nhân (Multiplier) và tích số (Product) và có thể có thêm một số phần tử khác. Mỗi phần tử pin nói trên cần được nối với một phần tử hiển thị Hex Digit Display.
3. Số lượng mạch con nên từ 1 đến 3.

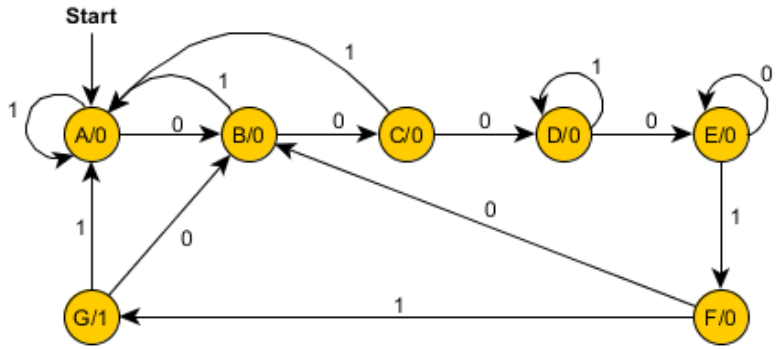
**Gợi ý:**

- Số nhân và số bị nhân là 8 bit, nhưng đơn vị ALU, kết quả nhân (Product) là 16 bit.
- Vì chúng ta thực hiện phép nhân số 8 bit bằng cách cộng nhiều lần số 8 bit, nên có thể sử dụng Adder 8 bit làm đơn vị "16-bit ALU" ở hình vẽ trên, với điều kiện là sau mỗi phép cộng, kết quả được dịch phải 1 vị trí, bit bậc 0 "bật ra" được cất đi, còn Carry_out (1 hoặc 0) được đưa vào vị trí bít cao nhất (bậc 7).
- Như vậy đơn vị "16-bit ALU" ở hình vẽ trên có thể bao gồm 1 Adder 8 bit và 2 Shiffter 8 bit bố trí liên tiếp ở đầu ra của Adder.

## Bài số 17: Xây dựng mạch điện thực hiện chức năng của máy trạng thái

Hãy xây dựng mạch điện thực hiện chức năng của một máy trạng thái (State Machine) như được mô tả ở hình vẽ sau.

Chú ý: S2 là bit có bậc cao nhất (bậc 2) và S0 là bit bậc thấp nhất (bậc 0). Cũng cần chú ý thêm là: khi cho một giá trị nhất định của input (điều kiện chuyển trạng thái), 3 bit mô tả trạng thái của hệ thống là S2S1S0 không tăng dần theo cách tăng của số nhị phân 3 bit. Khi xây dựng mạch điện, hãy ký hiệu (Label) input của hệ thống là "In". Đầu ra của máy trạng thái cần được ghi nhãn là Out. Máy trạng thái này là một máy kiểu "Moore state machine", bởi vì đầu ra (output) của nó chỉ phụ thuộc vào trạng thái hiện thời.




| | S2 | S1 | S0 |
|---|---|---|---|
| A = | 0 | 0 | 0 |
| B = | 1 | 0 | 0 |
| C = | 0 | 1 | 0 |
| D = | 0 | 0 | 1 |
| E = | 1 | 0 | 1 |
| F = | 0 | 1 | 1 |
| G = | 1 | 1 | 0 |

**<u>Gợi ý:</u>**

1. Bước đầu tiên: Lập bảng trạng thái (state table).
2. Bước tiếp theo: dựa trên bảng trạng thái để xây dựng mạch điện. Hãy theo các hướng dẫn sau:
   - Không tạo mạch con (subcircuit).
   - Sử dụng 3 D flip flops (sử dụng tham số Trigger đặt sẵn), 3 đầu ra của chúng sẽ nối với các phần tử pin (ouput), gán nhãn là S2, S1 và S0.
   - Đầu ra Out là đầu ra của một cổng AND có 3 đầu vào nối với 3 đầu ra của D flip flop mà chúng ta đã ký hiệu là S2, S1 và S0.
   - Cần một phần tử Clock (trong nhóm Wiring) nối với 3 đầu vào "Clock" của 3 D flip flop.

## Bài số 18 Viết chương trình nhân 2 số 8 bit không dấu

- Mô tả chương trình (đặt tên file là test02-multiplication.txt):
  - Input: Cho 2 số 8 bit, thí dụ 0FH (15) và 04H
  - Output: Tích của chúng (=60 = 3CH)
- Chú ý: Thực chất máy tính (mô phỏng) mà chúng ta đã xây dựng là máy tính 16 bit (các thanh ghi và ô nhớ đều có kích thước 16 bit. Tuy nhiên để bài tập không quá phức tạp khi phải xử lý tình huống kết quả phép nhân là số lớn hơn 16 bit, nên sinh viên chỉ được yêu cầu Input là số 8 bit. Khi làm bài tập này sinh viên có thể thử các trường hợp tổng số bit của cả 2 số đưa vào không quá 16).

**Gợi ý:**

- Hãy đọc lại kỹ mục "12.4. Kiểm tra hệ thống máy tính" trong tài liệu này. Trong mục này đã phân tích rất kỹ lưỡng việc viết và cho thi hành một chương trình đơn giản là cộng 2 số.
- Với bài tập này, có thêm một số điều cần chú ý, điều đầu tiên cần chú ý, đó là tập vĩ chỉ thị (mức máy thông thường) không có chỉ thị nhân, chính vì vậy chúng ta chỉ có thể sử dụng chỉ thị cộng ADDD hoặc ADDL.
- Trước khi vào vòng lặp, chương trình của chúng ta sẽ kiểm tra xem số nhân có bằng 0 hay không, nếu đúng thì gán 0 cho Product rồi dừng chương trình; Sau đó sẽ kiểm tra số nhân có bằng 1 hay không, nếu đúng thì gán cho Product bằng số bị nhân rồi dừng chương trình.
- Chúng ta sẽ viết một vòng lặp, số lần lặp bằng giá trị của số nhân trừ đi 1, trong vòng lặp sẽ có phép cộng số bị nhân với chính nó. Ở bài tập đơn giản này, chưa yêu cầu tối ưu hóa chương trình (thí dụ: tính số lần lặp theo số nhỏ hơn giữa số bị nhân và số nhân).
- Tập vĩ chỉ thị cũng không có các chỉ thị kiểu chu trình như LOOP hoặc REP (repeat) trong nhiều ngôn ngữ assembly, vì vậy chúng ta buộc phải sử dụng một trong các chỉ thị nhảy có điều kiện (JPOS, JZER, JNEG, JNZE) kết hợp với việc sử dụng một con đếm số vòng lặp và tự chúng ta phải tăng/giảm con đếm sau mỗi vòng lặp.
- Khi thử, hãy đưa vào các số nhỏ, nhất là số nhân (Multiplier) để chương trình hoàn thành trong thời gian ngắn, vì chương trình của chúng ta thực hiện phép nhân bằng cách lặp lại nhiều lần phép cộng.

## Bài số 19 Viết chương trình chia (nguyên) 2 số 16 bit không dấu.

Mô tả chương trình (đặt tên file là test03-division.txt):

- Input: Cho 2 số 16 bit, số thứ nhất là số bị chia (dividend), số thứ 2 là số chia (divisor), thí dụ 0FH (15) và 03H
- Output: Thương (quotient) của chúng (05)

**Gợi ý:**

- Hãy đọc lại kỹ mục "12.4. Kiểm tra hệ thống máy tính" trong tài liệu này. Trong mục này đã phân tích rất kỹ lưỡng việc viết và cho thi hành một chương trình đơn giản là cộng 2 số.
- Với bài tập này, có thêm một số điều cần chú ý, điều đầu tiên cần chú ý, đó là tập vĩ chỉ thị (mức máy thông thường) không có chỉ thị chia, chính vì vậy chúng ta chỉ có thể sử dụng chỉ thị trừ SUBD hoặc SUBL.
- Chúng ta sẽ viết một vòng lặp trong đó có việc lấy số bị chia trừ đi số chia. Trước khi vào vòng lặp, chương trình của chúng ta sẽ kiểm tra xem số chia bằng 0 hay không, nếu đúng thì dừng chương trình; Sau đó sẽ kiểm tra số bị chia có nhỏ hơn số chia hay không, nếu đúng thì gán cho thương bằng 0.
- Chúng ta viết vòng lặp kiểu "Repeat … Until" trong ngôn ngữ lập trình Pascal, trong vòng lặp thực hiện lấy số bị chia trừ đi số chia, nếu kết quả bằng 0 hoặc nhỏ hơn số chia thì kết thúc vòng lặp. Chúng ta sẽ ghi số lần lặp vào một biết đếm, khi kết thúc vòng lặp giá trị của biến đếm chính là kết quả của phép chia nguyên.
- Tập vĩ chỉ thị cũng không có các chỉ thị kiểu chu trình như LOOP hoặc REP (repeat) trong nhiều ngôn ngữ assembly, vì vậy chúng ta buộc phải sử dụng một trong các chỉ thị nhảy có điều kiện (JPOS, JZER, JNEG, JNZE) kết hợp với việc sử dụng một con đếm số vòng lặp và tự chúng ta phải tăng/giảm con đếm sau mỗi vòng lặp.
- Khi thử, hãy đưa vào các số nhỏ, nhất là số bị chia để chương trình hoàn thành trong thời gian ngắn, vì chương trình của chúng ta thực hiện phép chia bằng cách lặp lại nhiều lần phép trừ.
- Máy tính mô phỏng của chúng ta có thể tính với số bị trừ lớn nhất là 32767 = 7FFFH.

## Bài số 20: Viết chương trình kiểm tra số nguyên tố có chương trình con

Yêu cầu cụ thể:

- Kiểm tra một số lớn hơn 3 cho trước có phải là số nguyên tố hay không.
  - Input: Cho một số 16 bit có giá trị nằm trong miền 4..32767 (7FFFH).
  - Output: True (1) hoặc False (0)
- Chương trình chính có lời gọi CALL tới chương trình con.
- Chương trình con có sử dụng các chỉ thị (mức máy thông thường) thực hiện phép cộng hoặc trừ, kết thúc chương trình con là chỉ thị RETN.
- Đặt tên file là test04-call-subroutine.txt

**Gợi ý:**

- Hãy đọc lại kỹ mục "12.4. Kiểm tra hệ thống máy tính" trong tài liệu này. Trong mục này đã phân tích rất kỹ lưỡng việc viết và cho thi hành một chương trình đơn giản là cộng 2 số.
- Với bài tập này, có thêm một số điều cần chú ý, điều đầu tiên cần chú ý, đó là tập vĩ chỉ thị (mức máy thông thường) không có chỉ thị chia, chính vì vậy chúng ta chỉ có thể sử dụng chỉ thị trừ SUBD hoặc SUBL.
- Tập vĩ chỉ thị cũng không có các chỉ thị kiểu chu trình như LOOP hoặc REP (repeat) trong nhiều ngôn ngữ assembly, vì vậy chúng ta buộc phải sử dụng một trong các chỉ thị nhảy có điều kiện (JPOS, JZER, JNEG, JNZE) kết hợp với việc sử dụng một con đếm số vòng lặp và tự chúng ta phải tăng/giảm con đếm sau mỗi vòng lặp.
- Chương trình chính sẽ có một vòng lặp trong đó có việc gọi chương trình con. Chương trình chính truyền vào chương trình con số cần kiểm tra (số bị chia), ký hiệu là N và số chia là một con số có giá trị từ 2 đến (N div 2) nếu (N div 2) không còn dư hoặc (N div 2) +1 trong trường hợp ngược lại.
- Chương trình con sẽ lấy số bị chia (dividend) và số chia (divisor) được truyền vào từ chương trình chính làm số bị trừ (minuend) và số trừ (subtrahend). Việc trừ được thực hiện liên tiếp cho đến khi hiệu số (difference) nhỏ hơn hoặc bằng 0 thì dừng:
  - Nếu hiệu số bằng 0, nghĩa là số bị chia chia hết cho số chia, suy ra số được kiểm tra không phải là số nguyên tố.
  - Nếu hiệu số là âm, nghĩa là số bị chia không chia hết cho số chia, suy ra số được kiểm tra có thể là số nguyên tố.
- Nếu chương trình chính thực hiện xong vòng lặp mà không có lần lặp nào phát hiện thấy số cần kiểm tra không phải là số nguyên tố, thì số cần kiểm tra là số nguyên tố.
- Khi thử, hãy bắt đầu thử với các số nhỏ để chương trình hoàn thành trong thời gian ngắn, vì chương trình của chúng ta thực hiện phép chia bằng cách lặp lại nhiều lần phép trừ.
- Máy tính mô phỏng của chúng ta có thể kiểm tra với số lớn nhất là 32767 = 7FFFH.
- Hãy thử chương trình của bạn với một số số nguyên tố sau: 5, 7, 11(00BH), … , 223(0DFH), 227(0E3H), … , 487(01E7H), 491(01EBH), 499(01F3H).