Kiểm tra giữa kì

#1 - Trình bày khái niệm Kiến trúc máy tính và các thành phần của Kiến trúc máy tính

- Kiến trúc máy tính (Computer architecture) là một khoa học về lựa chọn và kết nối các thành phần phần cứng của máy tính nhằm đạt được các yêu cầu
  - Hiệu năng/tốc độ (performance): nhanh
  - Chức năng (functionality): nhiều tính năng
  - Giá thành (cost): rẻ
- Kiến trúc máy tính là một trong hai khái niệm cơ bản của công nghệ máy tính
- Kiến trúc máy tính gồm 3 thành phần cơ bản là
  - Kiến trúc tập lệnh (Instruction set architecture - ISA)
  - Vi kiến trúc (micro-architecture)
  - Thiết kế hệ thống (System Desgin)

#2 - Trình bày các thanh ghi của vi xử lý Intel 8086

Thanh ghi đoạn (Segment Register)

- Các thanh ghi đoạn trong CPU giống như các chỉ dẫn đường, giúp CPU biết phải tìm dữ liệu ở đâu trong bộ nhớ
  - Thanh ghi đoạn mã CS (Code - Segment) chỉ cho CPU biết chương trình đang thực thi nằm ở đâu
  - Thanh ghi đoạn dữ liệu DS (Data - Segment) cùng với thanh ghi đoạn dữ liệu phụ ES (Extra Segment) cho biết dữ liệu cần xử lý nằm ở đâu
  - Thanh ghi đoạn ngăn xếp SS (Stack - Segment) được sử dụng để lưu các giá trị trung gian hoặc các địa chỉ trở về khi gọi hàm

Thanh ghi đa năng

- Trong khối EU có bốn thanh ghi đa năng 16 bit là AX, BX, CX, DX. Khi cần chứa các dữ liệu 8 bít thì mỗi thanh ghi có thể tách ra thành hai thanh ghi 8 bit cao và thấp để làm việc độc lập
  - AX (accumulator): Thanh ghi tích lũy
  - BX (base): thanh ghi cơ sở, thường được dùng để chứa địa chỉ ô nhớ
  - CX (counter): Sử dụng làm bộ đếm số vòng lặp
  - DX (Data): Thanh ghi dữ liệu

Các thanh ghi con trỏ và chỉ số

- IP (Instruction Pointer): Con trỏ lệnh. IP luôn trỏ vào lệnh tiếp theo sẽ được thực hiện nằm trong đoạn mã CS (Code - Segment)

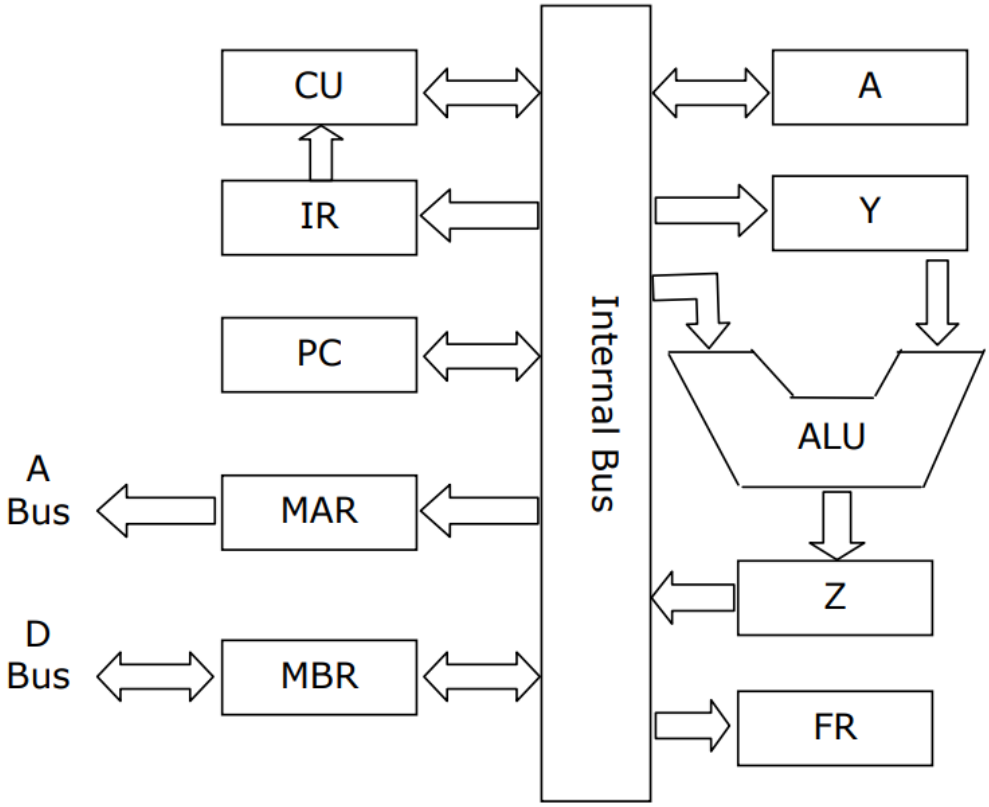
- BP (Base Pointer): Con trỏ cơ sở - sử dụng với đoạn ngăn xếp
- SP (Stack Pointer): Con trỏ ngăn xếp. SP luôn chứa địa chỉ đỉnh ngăn xếp
- SI (Source Index): Thanh ghi chỉ số nguồn. SI thường dùng chứa địa chỉ ô nhớ nguồn trong các thao tác chuyển dữ liệu
- DI (Destination Index): Thanh ghi chỉ số đích. DI thường dùng chứa địa chỉ ô nhớ đích trong các thao tác chuyển dữ liệu

#3 - Trình bày khuôn dạng lệnh của vi xử lý Intel 8086

- Trong chương trình hợp ngữ, mỗi lệnh được đặt trên một dòng - dòng lệnh. Do đó, ta không cần phải ngăn cách mỗi lệnh bằng dấu `;`
- Lệnh có 2 dạng
  - Lệnh thật (VD: MOV, SUB, ADD,..) khi dịch, lệnh gợi nhớ được dịch ra mã máy
  - Lệnh giả: là các hướng dẫn chương trình dịch (VD: MAIN PROC, .DATA, END MAIN,...) khi dịch, lệnh giả không được dịch ra mã máy mà chỉ có tác dụng định hướng cho chương trình dịch
- Không phân biệt chữ hoa hay chữ thường trong các dòng lệnh hợp ngữ
- Cú pháp của một lệnh Assembly thường có dạng sau:

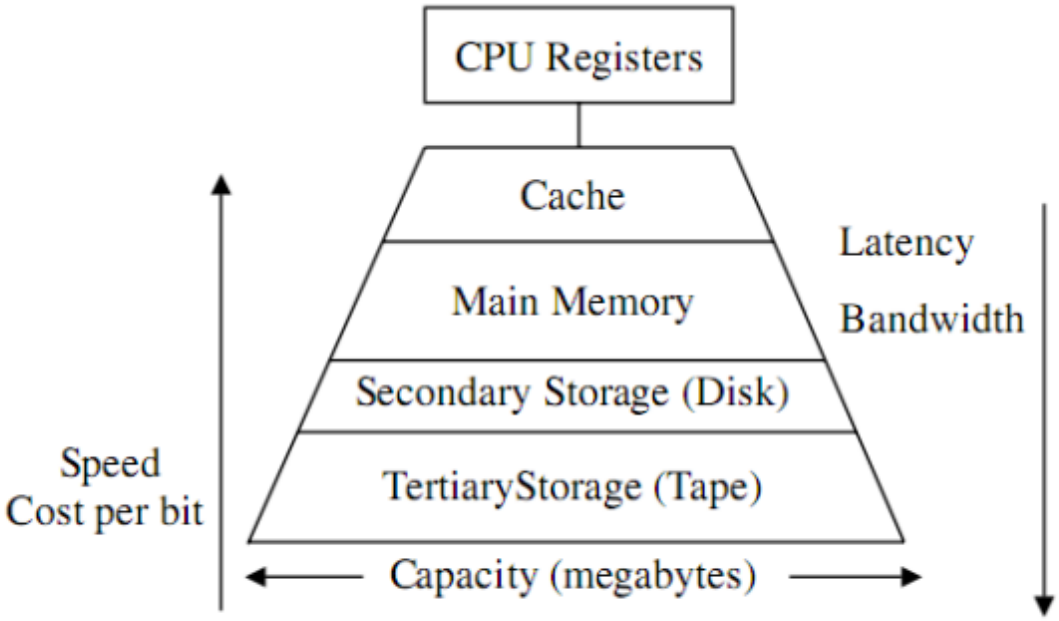
  `[label] [mnemonic] [operands] ; [comment]`
- Trong đó
  - **label** (Trường tên): chứa các nhãn, tên biến, hoặc tên thủ tục, là tên đặt cho một vị trí nhất định trong chương trình. Nhãn giúp chúng ta dễ dàng tham chiếu đến các vị trí này khi cần thiết
  - **mnemonic**: là tên của lệnh (hoặc mã lệnh) cần thực hiện. Trong trường mã lệnh sẽ có lệnh thật hoặc lệnh giả. Đối với các lệnh thật thì trường này chứa các mã lệnh gợi nhớ. Mã lệnh này sẽ được chương trình dịch ra mã máy. Đối với các hướng dẫn chương trình dịch thì trường này chứa các lệnh giả và sẽ không được dịch ra mã máy
  - **operands**: (Trường toán hạng) là các tham số của lệnh
  - **comment**: là phần chú thích, bắt đầu bằng dấu chấm phẩy `;`

#4 - Trình bày cấu trúc của bộ xử lý trung tâm



- Các thành phần của CPU theo sơ đồ này bao gồm
  - Bộ điều khiển (Control Unit - CU)
  - Bộ tính toán số học và logic (Arithmetic and Logic Unit)
  - Bus trong CPU (CPU Internal Bus)
  - Các thanh ghi của CPU
    - Thanh ghi tích lũy A (Accummulator)
    - Bộ đếm chương trình PC (Program Counter)
    - Thanh ghi lệnh IR (Instruction Register)
    - Thanh ghi địa chỉ bộ nhớ MAR (Memory Address Register)
    - Thành ghi đệm dữ liệu MBR (Memory Buffer Register)
    - Các thành ghi tạm thời Y và Z
    - Thanh ghi cờ FR (Flag Register)

#5 - Trình bày hệ thống bộ nhớ phân cấp trong các hệ thống máy tính



Hình 33 Cấu trúc phân cấp hệ thống nhớ

- Trong cấu trúc phân cấp hệ thống nhớ, dung lượng các thành phần tăng theo chiều từ các thanh ghi của CPU đến bộ nhớ ngoài
- Ngược lại, tốc độ truy nhập hay băng thông và giá thành tăng theo chiều từ bộ nhớ ngoài đến thanh ghi của CPU
- Như vậy, các thanh ghi của CPU có dung lượng nhỏ nhất nhưng có tốc độ truy cập nhanh nhất và cũng có giá thành cao nhất. Bộ nhớ ngoài có dung lượng lớn nhất, nhưng tốc độ truy cập thấp nhất. Bù lại, bộ nhớ ngoài có giá thành rẻ nên có thể được sử dụng với dung lượng lớn
- **CPU Registers** (các thanh ghi của CPU)
  - Dung lượng rất nhỏ, khoảng từ vài chục bytes đến vài KB
  - Tốc độ truy nhập rất cao (thời gian truy nhập khoảng 0.25ns)
  - Giá thành đắt
  - Sử dụng để lưu toán hạng đầu vào và kết quả đầu ra của các lệnh
- **Cache** (bộ nhớ Cache - còn được gọi là "bộ nhớ thông minh")
  - Dung lượng tương đối nhỏ, khoảng 64KB đến 32KB
  - Tốc độ truy nhập cao (thời gian truy nhập khoảng 1-5ns)
  - Giá thành đắt
  - Sử dụng để lưu lệnh và dữ liệu cho CPU xử lý
- **Main memory** (bộ nhớ chính)
  - Gồm ROM và RAM, có kích thước khá lớn (với hệ thống 32 bit, dung lượng khoảng 256MB - 4GB)
  - Tốc độ truy nhập chậm (thời gian truy nhập khoảng 50-70ns)
  - Giá thành tương đối rẻ
  - Sử dụng để lưu lệnh và dữ liệu của hệ thống và của người dùng

- **Secondary memory** (bộ nhớ ngoài)
  - Có dung lượng rất lớn (khoảng từ 20GB - 1000GB)
  - Tốc độ truy nhập rất chậm (thời gian truy nhập khoảng 5ms)
  - Giá thành rẻ
  - Sử dụng để lưu dữ liệu lâu dài dưới dạng các tệp (files)
- **Vai trò của việc phân cấp hệ thống bộ nhớ**
  - Tăng hiệu năng hệ thống: Thời gian trung bình CPU truy nhập dữ liệu từ hệ thống tiệm cận thời gian truy nhập cache -> Dung hòa được CPU có tốc độ cao và phần bộ nhớ chính, bộ nhớ ngoài có tốc độ thấp
  - Giảm giá thành sản xuất
    - Các thành phần đắt tiền (thanh ghi và cache) được sử dụng với dung lượng nhỏ
    - Các thành phần rẻ tiền hơn (bộ nhớ chính và bộ nhớ ngoài) được sử dụng với dung lượng lớn

#6 - Phân biệt hai loại kiến trúc máy tính CISC và RISC

- CISC (Complex Instruction Set Computer) và RISC (Reduced Instruction Set Computer) là hai loại kiến trúc máy tính phổ biến
- **CISC (Complex Instruction Set Computer)**
  - CISC có một tập hợp lệnh phức tạp và đang dạng
  - CISC sử dụng định dạng biến đổi từ 16-64 bit cho mỗi lệnh
  - CISC có thể thực hiện nhiều hoạt động trong một lệnh duy nhất
  - CISC giảm kích thước chương trình và do đó yêu cầu ít chu kỳ bộ nhớ hơn để thực hiện các chương trình
- **RISC (Reduced Instruction Set Computer)**
  - RISC sử dụng một tập hợp lệnh nhỏ và đơn giản
  - RIS sử dụng định dạng cố định (32 bit) và hầu hết là lệnh dựa trên đăng ký
  - Mỗi lệnh RISC thực hiện chỉ một hoạt động và CPU có thể thực hiện chúng trong một chu kỳ
  - RISC yêu cầu bộ nhớ nhanh để lưu trữ các lệnh
- Trường hợp sử dụng
  - CISC thường được sử dụng
    - Khi cần giảm kích thước mã: CISC sử dụng các lệnh phức tạp có thể thực hiện nhiều hoạt động, giảm số lượng mã cần để thực hiện một tác vụ
    - Khi cần hiệu quả về bộ nhớ
    - Khi sử dụng phần mềm đã có

    > -> CISC được sử dụng rộng rãi trong các máy tính cá nhân và máy chủ. Kiến trúc x86 và x64, mà hầu hết các hệ điều hành hiện đại như Windows, macOS, và nhiều phiên bản của Linux đều chạy trên đó

  - RISC thường được sử dụng

- Khi cần thực hiện nhanh các lệnh
- Khi cần tiết kiệm năng lượng: RISC tiêu thụ ít năng lượng hơn CISC, làm cho chúng lý tưởng cho các thiết bị di động
- Khi cần viết phần mềm hiệu quả

> -> RISC thường được sử dụng trong các thiết bị di động và nhúng như điện thoại thông minh, và máy tính bảng do tiêu thụ năng lượng thấp hơn

#7 - Trình bày 2 nguyên lý tạo ra cơ chế hoạt động thông minh của bộ nhớ cache

- Cache hoạt động dựa trên hai nguyên lý cơ bản
  - **Nguyên lý lân cận về không gian (Spatial locality)**: "Nếu một ô nhớ đang được truy nhập thì xác suất các ô nhớ liền kề với nó được truy nhập trong tương lai gần là rất cao"
  - **Nguyên lý lân cận về thời gian (Temporal locality)**: "Nếu một ô nhớ đang được truy nhập thì xác suất nó được truy nhập lại trong tương lai là rất cao"
- Nhờ hai nguyên lý này, bộ nhớ cache có thể giảm đáng kể thời gian truy cập dữ liệu, tăng hiệu suất hoạt động của hệ thống

#8 - Nêu các địa chỉ của lệnh trong vi xử lý 8086. Cho một ví dụ minh họa mỗi dạng lệnh đó

- Chế độ địa chỉ (Addressing mode) là cách để CPU tìm thấy toán hạng cho các lệnh của nó khi hoạt động

1. Chế độ địa chỉ thanh ghi (Register Addressing Mode)

- Trong chế độ địa chỉ này, người ta dùng các thanh ghi bên trong CPU như là các toán hạng để chứa dữ liệu cần thao tác
- Việc thực hiện lệnh có thể đạt tốc độ truy nhập cao hơn so với các lệnh có truy nhập đến bộ nhớ
- Ví dụ: `MOV BX, DX` chuyển nội dung DX vào BX

2. Chế độ địa chỉ tức thì (immediate Addressing Mode)

- Dữ liệu được truyền trực tiếp vào lệnh
- Ví dụ: `MOV AX, 5` . Trong đó, 5 là giá trị được di chuyển vào thanh ghi AX

3. Chế độ địa chỉ trực tiếp (Direct Addressing Mode): Địa chỉ của dữ liệu được cung cấp trực tiếp trong lệnh

- Ví dụ: `MOV AX, (1234H)`. Trong đó, dữ liệu được lưu tại địa chỉ bộ nhớ 1234H được di chuyển vào thanh ghi AX

4. Chế độ gián tiếp qua thanh ghi (Register Indirect Addressing Mode): Địa chỉ của dữ liệu được lưu trong một thanh ghi

- Ví dụ: `MOV AL, (BX)` -> chuyển ô nhớ có địa chỉ DS:BX vào AL

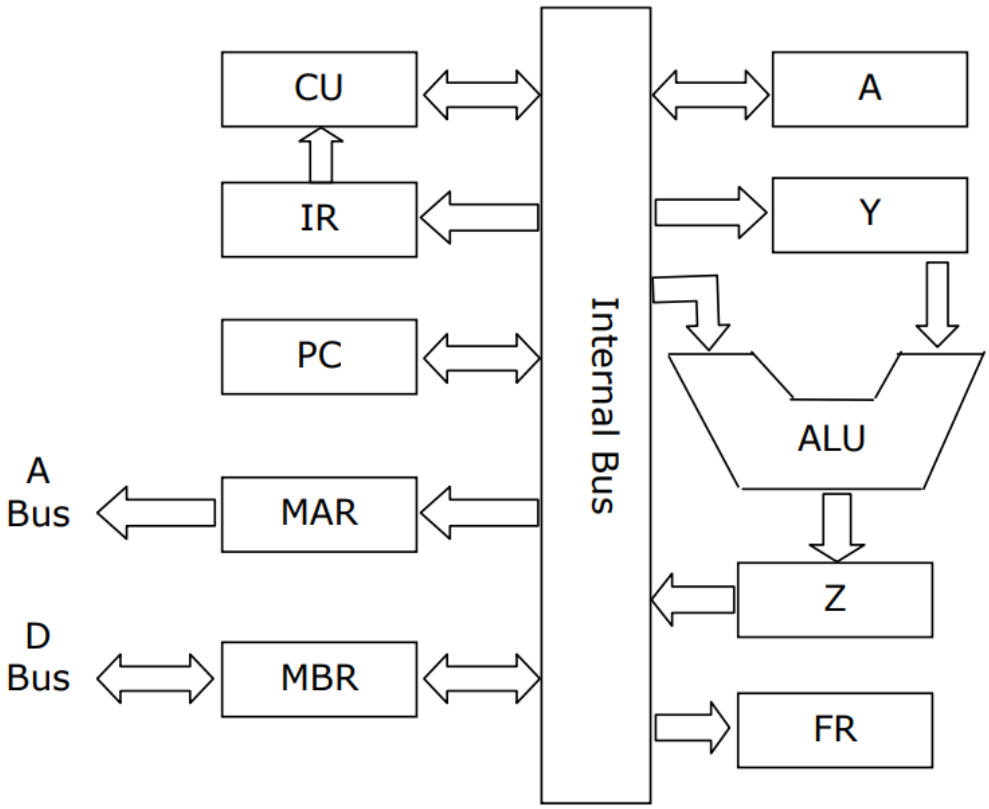
5. Chế độ địa chỉ tương đối cơ sở (Indexed Relative Addressing Mode): Địa chỉ của dữ liệu được tính dựa trên một thanh ghi cơ sở

- Ví dụ: `MOV AX, [BP + 10]`. Trong trường hợp này, dữ liệu được lấy từ vị trí bộ nhớ lưu trong BP cộng thêm 10

6. Chế độ địa chỉ tương đối chỉ số cơ sở (Based Indexed Relative Addressing Mode): Sử dụng cả thanh ghi cơ sở lẫn thanh ghi chỉ số để tính địa chỉ của toán hạng. Chế độ địa chỉ này hữu ích khi làm việc với mảng hai chiều

- Ví dụ: `MOV AX, [BX + SI + 20]` . Trong đó, AX nhận giá trị từ ô nhớ có địa chỉ bằng tổng của BX và SI và số 20

#9 - Vẽ sơ đồ khối tổng quát và trình bày chu trình xử lý lệnh của CPU



> Khoảng thời gian để CPU thực hiện xong một lệnh kể từ khi CPU cấp phát tín hiệu địa chỉ ô nhớ chứa lệnh đến khi nó hoàn tất việc thực hiện lệnh được gọi là *chu kỳ lệnh* (Instrution Cycle)

- **Chu trình xử lí lệnh** 1. Khi một chương trình được kích hoạt, hệ điều hành (OS - Operating System) nạp mã chương trình vào bộ nhớ trong 2. Địa chỉ của ô nhớ chứa lệnh của chương trình được nạp vào bộ đếm chương trình (Program Counter) 3. Địa chỉ ô nhớ chứa lệnh từ PC được chuyển đến bus địa chỉ thông qua thanh ghi MAR 4. Bus địa chỉ chuyển địa chỉ ô nhớ đến đơn vị quản lý bộ nhớ (MMU - Memory Management Unit) 5. MMU chọn ra ô nhớ và thực hiện lệnh đọc nội dung ô nhớ để

chuyển tiếp đến thanh ghi MBR 6. MBR chuyển lệnh đến thành ghi lệnh IR; IR chuyển lệnh vào bộ điều khiển CU 7. CU giải mã lệnh và sinh ra các tín hiệu điều khiển cần thiết, yêu cầu các bộ phận chức năng của CPU, như ALU thực hiện lệnh 8. Giá trị địa chỉ trong bộ đến PC được tăng lên 1 đơn vị lệnh và nó trỏ đến địa chỉ của ô nhớ chứa lệnh tiếp theo .

Bài tập

1. Cho một máy tính có độ rộng bus dữ liệu là 32 bit, quản lý được bộ nhớ có dung lượng tối đa là 64GB, bộ nhớ cache có dung lượng 4MB với dòng cache 64KB. Hãy xác định các thành phần địa chỉ của bộ nhớ khi sử dụng phương pháp ánh xạ trực tiếp.

- 64KB = 2^16
- 4MB = 2^22
- 64GB = 2^36
- Sồ dòng trong cache: 2^22 / 2^16 = 2^6 dòng -> line = 6 bit
- Word = 16bit
- Dung lượng bộ nhớ: 64GB = 2^36, cần 36 bít địa chỉ tổng cộng để địa chỉ hóa các ô nhớ -> Tag = 36 - 16 - 6 = 14 bit

2. Cho một máy tính có độ rộng bus dữ liệu là 64 bit, quản lý được bộ nhớ có dung lượng tối đa là 128GB, bộ nhớ cache có dung lượng 4MB và 4 đường cache với dòng cache 32KB. Hãy xác định các thành phần địa chỉ của bộ nhớ khi sử dụng phương pháp ánh xạ tập kết hợp.

- 128GB = 2^37
- 4MB = 2^22
- 32KB = 2^15
- Word = 15 bit
- Số dòng trong cache = 2^22 / 2^15 = 2^7 dòng -> Set = log2(2^7 / 4 đường) = 5 bit
- Dung lượng bộ nhớ: 128GB = 2^37, cần 37 bít địa chỉ tổng cộng để địa chỉ hóa các ô nhớ -> Tag = 37 - 15 - 5 = 17 bit

#11 - Cơ chế ống lệnh (pipeline) của CPU thường gặp phải những vấn đề gì? Nêu ý nghĩa các lệnh dưới đây (Rl, R2 là các thanh ghi và các lệnh quy ước theo dạng LỆNH <ĐÍCH> <GỐC>) và đưa ra một hướng giải quyết xung đột dữ liệu trong pipeline khi thực hiện đoạn chương trình sau:

(1) LOAD R2, #200

(2) LOAD Rl, #1000

(3) STORE (R1), R2

(4) SUBSTRACT R2, #210

(5) ADD 1000, #10

(6) ADD R2, (R1)

Cơ chế ống lệnh (pipeline) của CPU thường gặp phải 3 vấn đề

- **Vấn đề xung đột tài nguyên**: Vấn đề xung đột tài nguyên xảy ra khi hệ thống không cung cấp đủ tài nguyền phần cứng phục vụ CPU thực hiện đồng thời nhiều lệnh trong cơ chế ống lệnh
- **Vấn đề tranh chấp dữ liệu**: Tranh chấp dữ liệu cũng là một trong các vấn đề lớn của cơ chế ống lệnh và tranh chấp dữ liệu kiểu đọc sau khi ghi (RAW - Read After Write) là dạng xung đột dữ liệu hay gặp nhất
- **Vấn đề nảy sinh do các lệnh rẽ nhánh**: Theo thống kê, tỷ lệ các lệnh rẽ nhánh trong chương trình khoảng 10-30%. Do lệnh rẽ nhánh thay đổi nội dung của bộ đếm chương trình, chúng có thể phá vỡ tiến trình thực hiện tuần tự các lệnh trong ống lệnh vì lệnh được thực hiện sau lệnh rẽ nhánh có thể không phải là lệnh liền sau nó mà là lệnh ở một vị trí khác. Như vậy do kiểu thực hiện gối đầu, các lệnh liền sau lệnh rẽ nhánh đã được nạp và thực hiện dở dang trong ống lệnh sẽ bị đẩy ra làm cho ống lệnh trống rỗng và hệ thống phải bắt đầu nạp mới các lệnh từ địa chỉ đích rẽ nhánh

Câu hỏi 3.9: Cho đoạn chương trình sau (R1, R2 là các thanh ghi):

Lệnh	Ý nghĩa	Chế độ	GT R2	
LOAD R2, #500	R2← 500	Tức thì (immediate)	500	
LOAD R1, #2000	R1← 2000	Tức thì (immediate)	500	
STORE (R1), R2	Lưu nội dung của thanh ghi R2 vào ô nhớ có địa chỉ chứa trong thanh ghi R1	Gián tiếp (Indirect)	500	
ADD 2000, #30	Lấy nội dung của ô nhớ có địa chỉ 2000 cộng với 30, kết quả lưu vào ô nhớ có địa chỉ 2000	Trực tiếp (Direct)	500	
SUBSTRACT R2, #15	R2←R2-15 : Lấy nội dung thanh ghi R2 trừ đi 15, kết quả lưu vào thanh ghi R2	Tức thì (immediate)	485	
ADD R2, (R1)	Lấy nội dung của thanh ghi R2 cộng với nội dung của ô nhớ có địa chỉ trong thanh ghi R1, kết quả lưu vào thanh ghi R2	Gián tiếp (Indirect)	1015	

- Sửa pipeline
  - Đây là xung đột dữ liệu kiểu đọc trước khi ghi
  - Hướng giải quyết: chèn lệnh no-op vào những chỗ xung đột

#13 - Nêu chức năng và phương thức hoạt động của con trỏ ngăn xếp SP (Stack Pointer). Thanh ghi cờ (hay thanh ghi trạng thái) của vi xử lý có chức

năng gì? Nêu ý nghĩa của các cờ nhớ (C), cờ không (Z), cờ dấu (S)

- Chức năng của con trỏ ngăn xếp là chỉ đến địa chỉ của đỉnh ngăn xếp (stack). Ngăn xếp là một cấu trúc dữ liệu tuyến tính hoạt động theo cơ chế LIFO (Last in, First out), nghĩa là phần tử cuối cùng được đưa vào là phần từ đầu tiên được lấy ra
- Phương thức hoạt động của con trỏ ngăn xếp
  - Khi một phần tử mới được đẩy vào ngăn xếp, con trỏ ngăn xếp tăng lên để trỏ đến vị trí của phần tử đó
  - Khi một phần tử được lấy ra khỏi ngăn xếp, con trỏ ngăn xếp giảm để trỏ đến phần tử tiếp theo ở đỉnh ngăn xếp
- Chức năng của thanh ghi cờ là lưu trữ các kết quả của các phép tính và điều kiện so sánh để quyết định những hoạt động tiếp theo của vi xử lý
- Ý nghĩa của các loại cờ
  - **ZF** (Zero Flag): -Cờ Zero, ZF=1 nếu kết quả=0 và ZF=0 nếu kết quả<>0
  - **SF** (Sign Flag): Cờ dấu, SF=1 nếu kết quả âm và SF=0 nếu kết quả dương
  - **CF** (Carry Flag): Cờ nhớ, CF=1 nếu có nhớ/mượn, CF=0 trong trường hợp khác

Code

1. Viết chương trình hợp ngữ nhập vào 1 chuỗi ký tự không quá 20 ký tự chỉ gồm các chữ cái và chữ số, in các chữ số ra màn hình

```
.model small
.stack 100
.data
    endl DB 10,13,'Ketquala',10,13,'$'
    str DB 20 dup('$')
.code
main proc

   mov ax, @data
   mov ds, ax

   mov ah, 10
   lea dx, str
   int 21h

   mov ah, 9
   lea dx, endl
   int 21h

   lea si, str + 2

   PrintDigit:
```

```
        mov al, [si]
        cmp al, '$'
        JE EndPrintDigit

        CMP al, '0'
        JL NotDigit

        CMP al, '9'
        JG NotDigit

        mov ah, 2
        mov dl, [si]
        int 21h

    NotDigit:
        inc si
        jmp PrintDigit

    EndPrintDigit:

    mov ah, 4ch
    int 21h


    main endp
end
```

2. Viết chương trình hợp ngữ Assembly cho phép nhập một chuỗi các ký tự, việc nhập kết thúc khi nhấn #

```
.model small
.stack 100
.data
    endl DB 10,13,'Ketquala',10,13,'$'
    str DB 20 dup('$')
.code
main proc

   mov ax, @data
   mov ds, ax
```

```
   Nhap:
       mov ah, 1
       int 21h

       cmp al, '#'
       JE KetThucNhap:

       cmp al, '#'
       JNE Nhap:

   KetThucNhap:

   mov ah, 4ch
   int 21h


   main endp
end
```

3. Viết chương trình hợp ngữ Assembly 8086 xuất hiện chuỗi “KTMT_2024” và in kết quả ra màn hình.

```
.model small
.stack 100
.data
    endl DB 10,13,'Ketquala',10,13,'$'
    str DB 'KTMT_2024$'
.code
main proc

   mov ax, @data
   mov ds, ax

   mov ah, 9
   lea dx, str
   int 21h

   mov ah, 4ch
   int 21h
```

```
   main endp
end
```

4. Viết chương trình hợp ngữ nhập vào 1 chuỗi ký tự không quá 15 ký tự, chuyển đổi các ký tự chữ thường thành chữ in và in chuỗi ký tự ra màn hình

```
.model small
.stack 100
.data
    endl DB 10,13,'Ketquala',10,13,'$'
    str DB 20 dup('$')
.code
main proc

   mov ax, @data
   mov ds, ax

   mov ah, 10
   lea dx, str
   int 21h

   mov ah, 9
   lea dx, endl
   int 21h

   lea si, str + 2

   InHoa:
       mov dl, [si]
       cmp dl, 'a'
       JL InRa
       cmp dl, 'z'
       JG InRa

       sub dl, 32
   InRa:
       mov ah, 2
       int 21h

       inc si
```

```
        cmp [si], '$'
        JNE InHoa

    mov ah, 4ch
    int 21h

    main endp
end
```

5. Tính tổng từ 1 đến 150

```
.model small
.stack 100
.data
    endl DB 10,13,'Ketquala',10,13,'$'
    str DB 'KTMT_2024$'
.code
main proc

    mov ax, @data
    mov ds, ax



    mov si, 150
    xor bx, bx
    lap:
       add bx, si
       dec si
       cmp si, 0
       jne lap

    mov ah, 9
    lea dx, endl
    int 21h

    mov ax, bx
    mov bx, 10
    xor si,si
    lap1:
```

```
      xor dx, dx
      div bx
      push dx
      inc si

      cmp ax, 0
      jne lap1

   lap2:
      pop dx
      add dx, '0'

      mov ah, 2
      int 21h

      dec si
      cmp si, 0
      jne lap2:


   mov ah, 4ch
   int 21h

   main endp
end
```