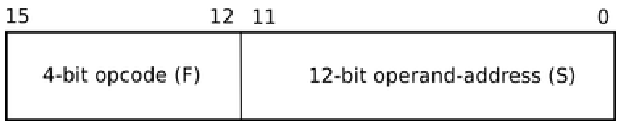
# Computer Architecture – Project #0

Implementation of MU0 Processor

## 1. Introduction

MU0 is a very simple and abstract design microprocessor. It has not been implemented but designed for study purpose at the university level (The name comes from the Manchester University). The study of MU0 design is really a good entry point for learning the processor designing.
MU0 is a 16-bit processor. This means that its ALU is capable on performing operations on 16-bit wide operands. And obviously, the accumulator register is a 16-bit register. Further, the instructions are also 16-bit, with a 12-bit address space. So, 8 kilobyte memory can be accessed. The MU0 instructions are fixed-format, fixed-length instruction with 1-operand address. Therefore, one of the operand and the destination of ALU operations is always implicit, which is accumulator register.
The MU0 implements total eight instructions in its instruction set. These instructions can be used to run some not-so-complex but useful programs.

MU0 Instruction Set Architecture



The op-code is 4-bit. These instructions are:

1. op-code 0000 (LDA S): load into accumulator, operand from memory location addressed as 'S'.
2. op-code 0001 (STO S): store the content of accumulator register at the memory location addressed as ''S'.
3. op-code 0010 (ADD S): ACC = ACC + [S] i.e. add content of memory location addressed as 'S with content of accumulator and save result at accumulator.
4. op-code 0011 (SUB S): ACC = ACC – [S]
5. op-code 0100 (JMP S): jump to the memory location addressed as 'S'. This will unconditionally change the program flow. PC = [S]
6. op-code 0101 (JGE S): if ACC>=0, then jump. This is conditional jump with condition that alu result is positive.
7. op-code 0110 (JNE S): another conditional jump with condition that ACC content is non-zero
8. op-code 0111 (STP): stop the program execution

* Note that 'S' mentioned in all but last instructions, is 12-bit LSB from instruction. The op-codes 1000 to 1111 are all reserved for future expansion. (all op-codes in binary format).

## 2. Assignment

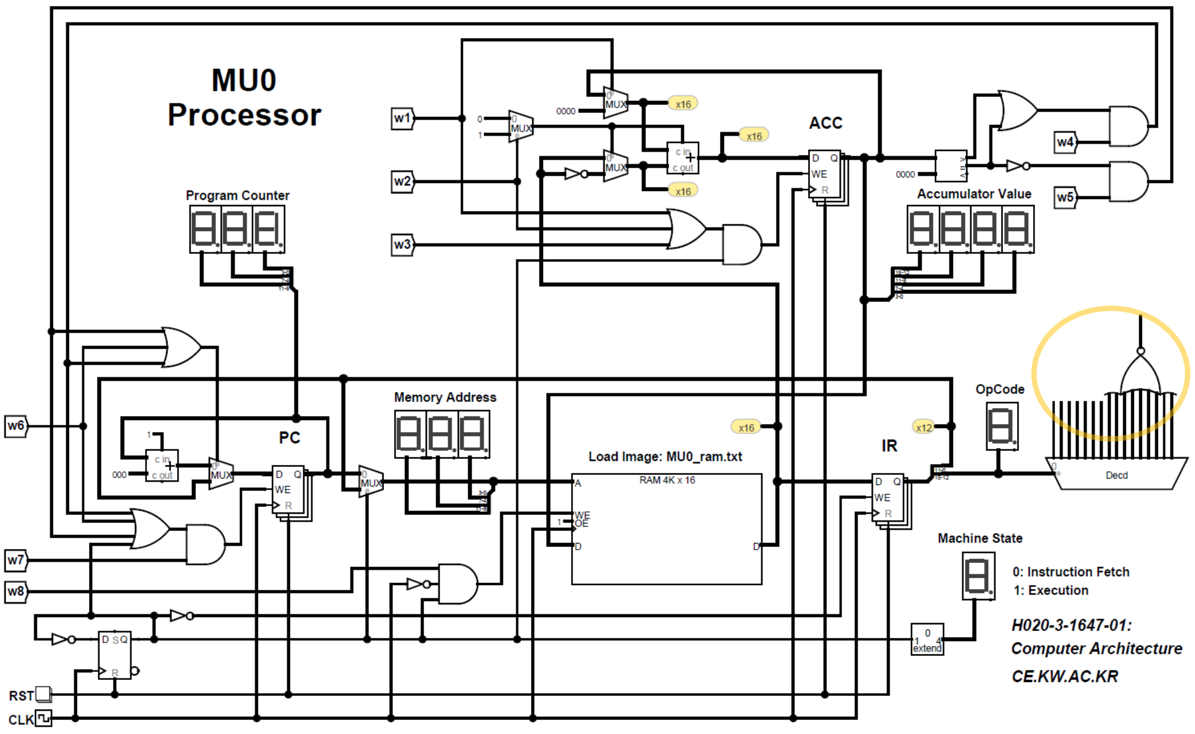
### 2.1 Practice for digital logics with Logisim-E

At first, you may need to be familiar with digital logics and Logisim-E simulator.
Follow the tutorials at the External reads in the Logisim-E github (https://github.com/logisim-evolution/logisim-evolution/blob/master/docs/docs.md).
Then, answer the question or implement the logics with constraints in the logisim-E simulator.

1. What is the Transmission gate?
2. Implement a MUX using Transmission gates
3. Implement a Latch using MUXs.
4. Implement D-FF using Latches.
5. Implement **Synchronous** Up/Down Counter using D-FFs and adders.
6. Compare Synchronous counter and Asynchronous counter such as a ripple counter.
7. What makes that Asynchronous counter is rarely used?

### 2.2 Complete the implementation of MU0 (2 cycle CPU)

This project is for understanding how to generate control signals by analyzing instructions. The following diagram shows the MU0 implementation with Logisim-E. The most of circuits are already designed, but the wires for control signals are intentionally dropped out and labeled as **w1** ~ **w8**.



As the instruction set shows the programmers point-of-view towards the processor architecture, from the above discussed instructions we can say that there are only two user-visible (programmer-visible) registers in MU0 viz. ACC (accumulator) and PC (program counter). In the implementation, however, you may notice another register IR (Instruction Register) for processor control.

In this implementation of MU0, the instructions are executed in two phases; FETCH and EXECUTE.

FETCH:

fetch the 16-bit instruction from memory into the instruction-register (IR) and then increment the PC content (PC = PC + 1). The fetched instruction goes to the instruction decoder (control unit) and appropriate control signal are set by decoder.

EXECUTE:

- get the operands required for processing instruction. Operand comes from memory for LDA, ADD and SUB instructions; and from instruction itself for all 3 branching (jump) instructions.
- carry-out the actual operation. ALU operation for ADD, SUB
- write-back the result to destination. to ACC for ADD, SUB; to memory for STO instructions; to PC for branch instructions.
- for STP instruction, nothing happens at the execute stage, but processor remains in same state forever (until reset).

■ In this MU0 project, you have to finish two tasks to operate the MU0 circuit of Logisim-E.
    - Complete the design by connecting all wires labeled as w1 to w8.
    - With the given binary file which will be loaded into the RAM, write the MU0 assembly program and explain how it operates.

**IMPORTANT NOTICE for Memory:**

Although the total memory size is 8K bytes, or 4K x 16 bit, there is no byte access mode. The memory in this implementation is a RAM, but practically some memory must be non-volatile to make it bootable and some must be volatile to store data dynamically. In this project we assume the memory from 0x000 to 0x7FF is non-volatile (i.e., ROM), and the memory form 0x800 to 0xFFF is volatile (i.e., RAM).

**3. 결과 Report (표지제외 최소 2장)**

- 문제의 해석 및 해결 방향
  - 각 회로에 대해 기능과 동작을 모두 상세히 설명할 것
    - 기능은 truth table 등을 사용할 것
    - 동작에 대해서는 각 변화에 대해 그림을 최대한 이용할 것
  - 실험 내용에 대한 설명

    ex) 자기가 구현한 하드웨어 구성과 어떠한 동작을 하는지 설명.
  - 문제점 및 개선점 기술
    - 각 입력에 대하여 어떻게 동작을 확인하였는지
    - Memory의 설계를 어떻게 바꾸면 위에서 언급한대로 동작하는지 논하면 추가점수

- 설계 의도와 방법
  - 구현한 회로도
  - 동작 검증을 위한 시나리오 제시 및 결과 분석
  - 구성한 회로 설명 필수
  - 시뮬레이션 결과와 프로그램에 의한 예상 결과 비교 분석

**4. 결과 Report Submission**

- Soft copy
  - Due data: **4월 7일 23:59 까지 (딜레이 받지 않음.)**
  - 결과 Report(pdf)와 프로젝트 폴더를 압축하여 U-campus에 upload.
  - **압축 파일명 양식: 학번_이름_Project_0.zip**
    - **ex) 2099722000_홍길동_ Project_0.zip**

**5. 공지 사항**

- 본 프로젝트는 개인 프로젝트임으로 문제 해결을 위한 협력은 금함.
- 부정행위 시 학칙에 의해 처벌될 예정