# HW5 Microprogrammed CPU 설계

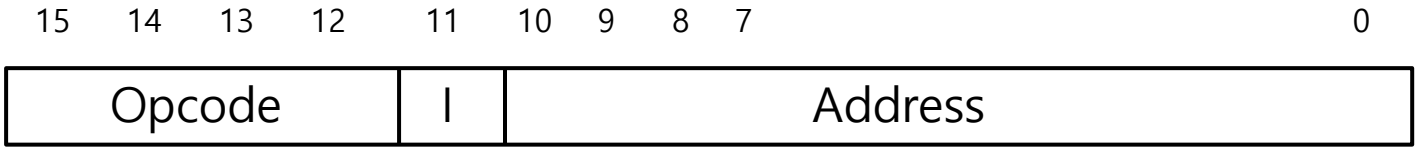
- Microprogramming 기법을 사용한 Mano CPU 설계
  - Mano CPU 를 Microprogrammed 방식으로 logisim 툴을 사용하여 설계
  - 입출력에 관련된 명령어들은 제외
  - 과제 2 Mano CPU의 제어신호 참조
  - 교재의 CPU 명령어 형식을 변경하고 메모리 참조 명령어 추가되었음

- 4 개의 테스트 프로그램

- 마감일: 12월 6일(화) 23시59분

# Mano CPU 명령어 변경

- new Instruction format
  - I bit 위치 : bit 15 -> bit 11
  - Opcode : 3bit -> 4bit
  - address field : 12 -> 11 bit




- 추가된 3개의 메모리 참조 명령어
  - SUB addr : AC <- AC – M[addr]
  - XCH addr : AC <-> M[addr]
  - BPA addr : if (AC>0) PC = addr

# Instruction Set

- Memory reference instruction

| Symbol | Hex Code | Description |
|---|---|---|
| AND | 0xxx | AC <- AC and DR |
| ADD | 1xxx | Add M to AC, carry to E |
| LDA | 2xxx | Load AC from M |
| STA | 3xxx | Store AC in M |
| BUN | 4xxx | Branch unconditionally to m |
| BSA | 5xxx | Save return address in m and branch to m+1 |
| ISZ | 6xxx | Increment M and skip if zero |
| | | |
| SUB | 9xxx | AC <- AC – M[EA] |
| XCH | Axxx | AC <-> M[EA] |
| | | |
| BPA | Cxxx | if (AC>0) PC = EA |



(* I/O Instruction 제외 *)

# Instruction Set

- Register reference instruction

| Symbol | Hex Code | Description |
|---|---|---|
| CLA | F400 | AC <- 0 |
| CLE | F200 | E <- 0 |
| CMA | F100 | Complement AC |
| CME | F080 | Complement E |
| CIR | F040 | Circulate right E and AC |
| CIL | F020 | Circulate left E and AC |
| INC | F010 | Increment AC |
| SPA | F008 | Skip if AC is positive |
| SNA | F004 | Skip if AC is negative |
| SZA | F002 | Skip if AC is zero |
| SZE | F001 | Skip if E is zero, |
| HLT | F800 | Halt computer |



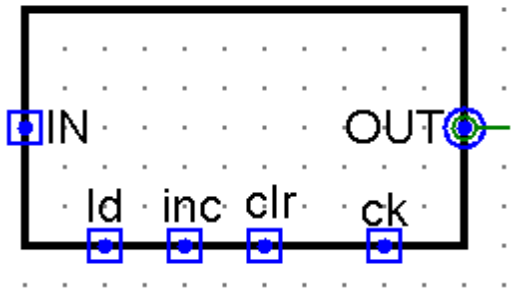
(* I/O Instruction 제외 *)

# Step-by-step Mano CPU 설계

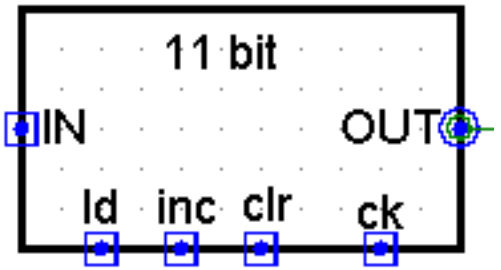
- Register 설계
- 버스 설계
- ALU 와 carry 제어 모듈 설계
  - 주어진 회로 사용
- Register reference instruction encoding
  - 명령어 실행주소 매핑을 위해 인코딩이 필요함
- Sequencer 설계
- 제어신호 배치와 Control memory 설계
- micro-coding

# Register 설계

- 16비트 레지스터 subcircuit
  - Memory lib의 counter 모듈 사용
  - load, increment, clear 기능이 있는 16 비트  레지스터




- 11 비트 레지스터 subcircuit
  - 버스에 연결이 쉽게 인터페이스는 16비트, 내부는 11비트

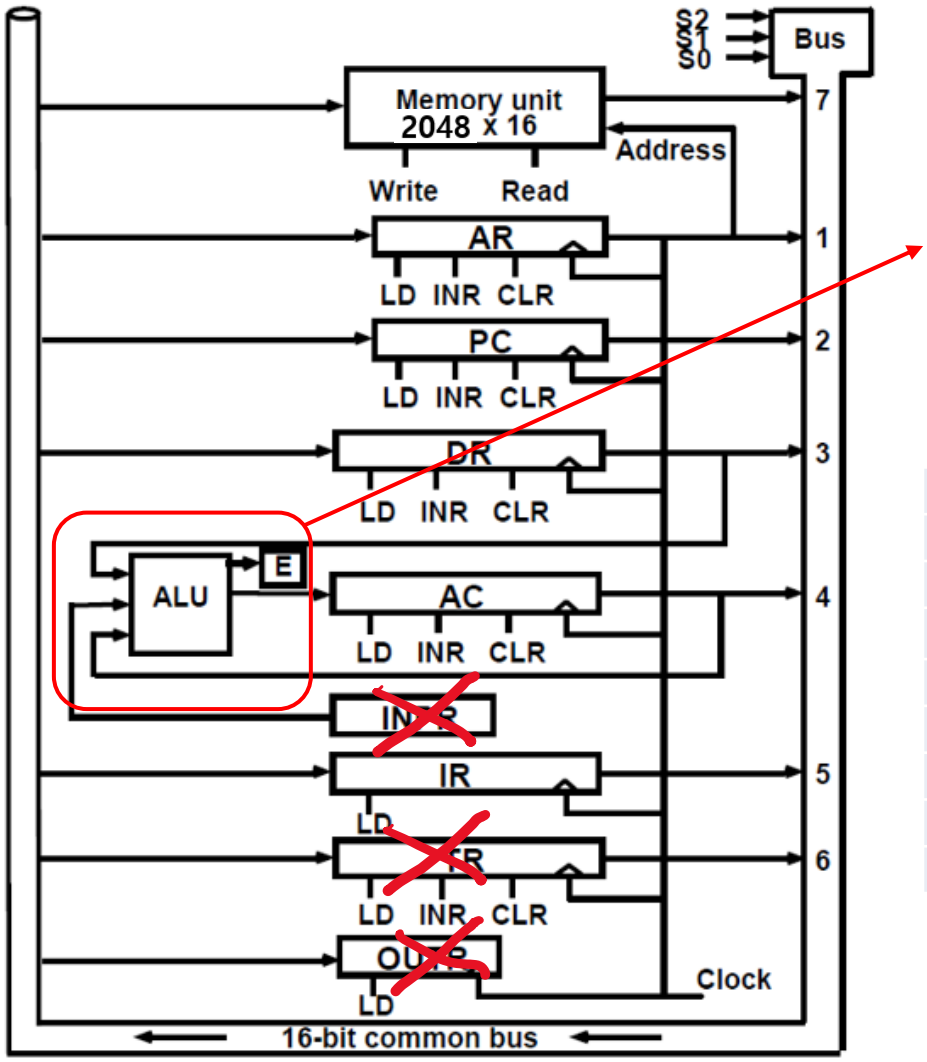


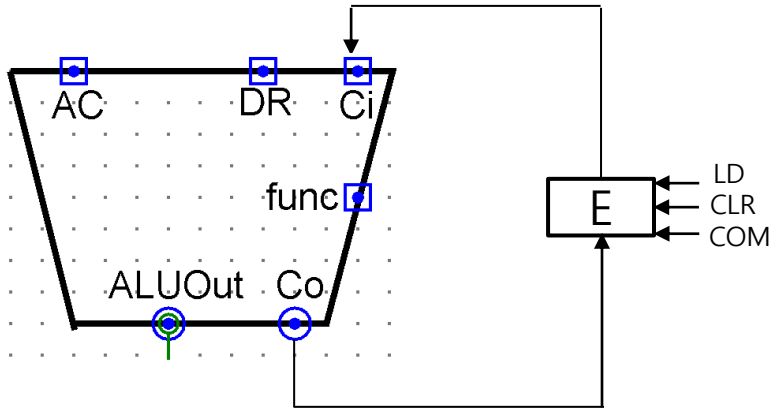
# Bus 설계

- Register 모듈을 사용하여 5개 레지스터 배치, MUX를 사용하여 버스에 연결
  - Main Memory :
    - address – 11 bits
    - data – 16 bits
    - data interface – separate load and store ports
  - AR
  - PC
  - DR
  - AC
  - IR
  - ~~TR~~ : 삭제

- 교재 p101 그림 5-4 참조

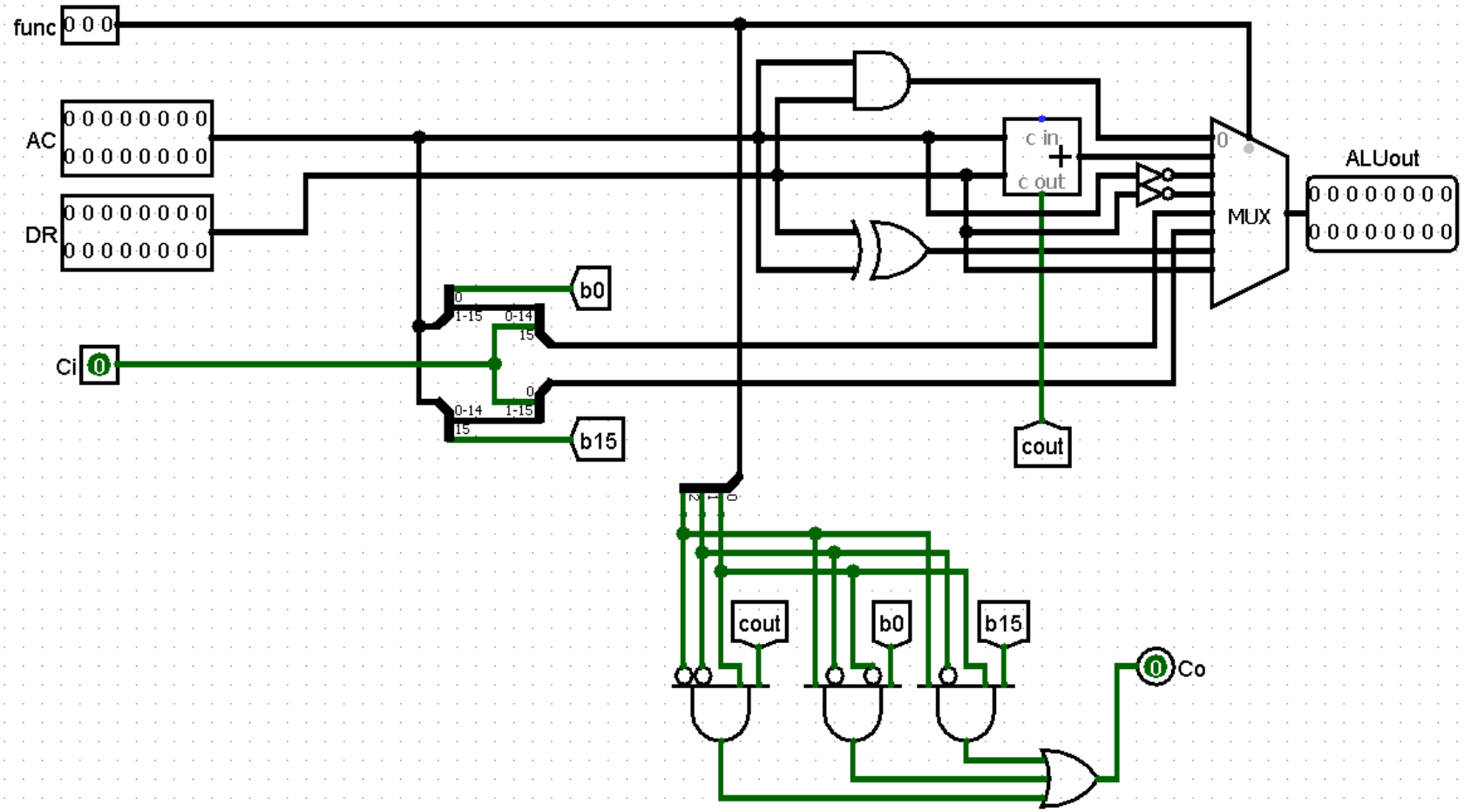
# ALU function



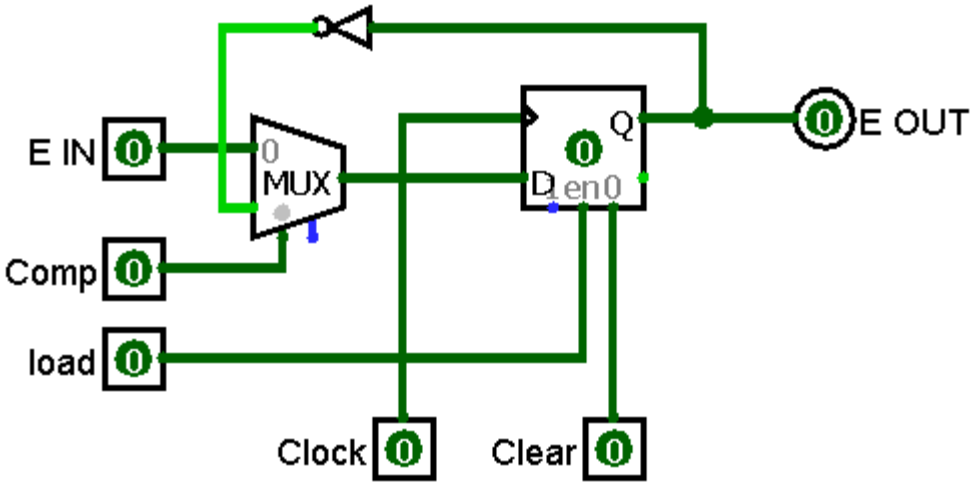

| Func Code | ALUOut |
|---|---|
| 000 | AND : AC ∩ DR |
| 001 | ADD : AC + DR |
| 010 | complement AC |
| 011 | Complement DR |
| 100 | CIR : Circular right E and AC |
| 101 | CIL : Circular left E and AC |
| 110 | XOR : AC ^ DR |
| 111 | pass DR |

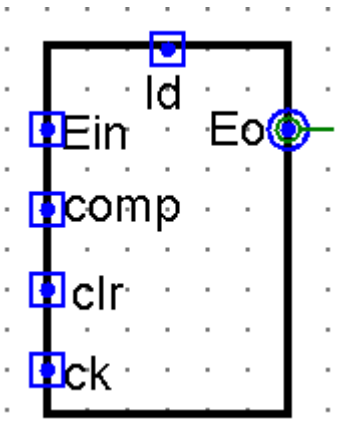
# ALU 회로


func
0 0 0
AC
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
DR
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
Ci
0
b0
b15
c in
c out
MUX
ALUout
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
cout
cout
b0
b15
Co
0
1-15
0-14
15

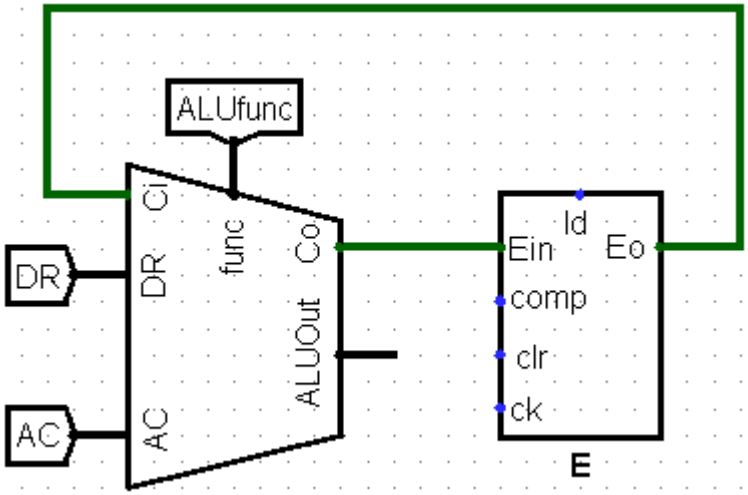
# carry 제어 모듈

- 기능
  - Ein & load : update carry with Ein
  - Comp & load : complement current carry
  - clear : asynchronous clear



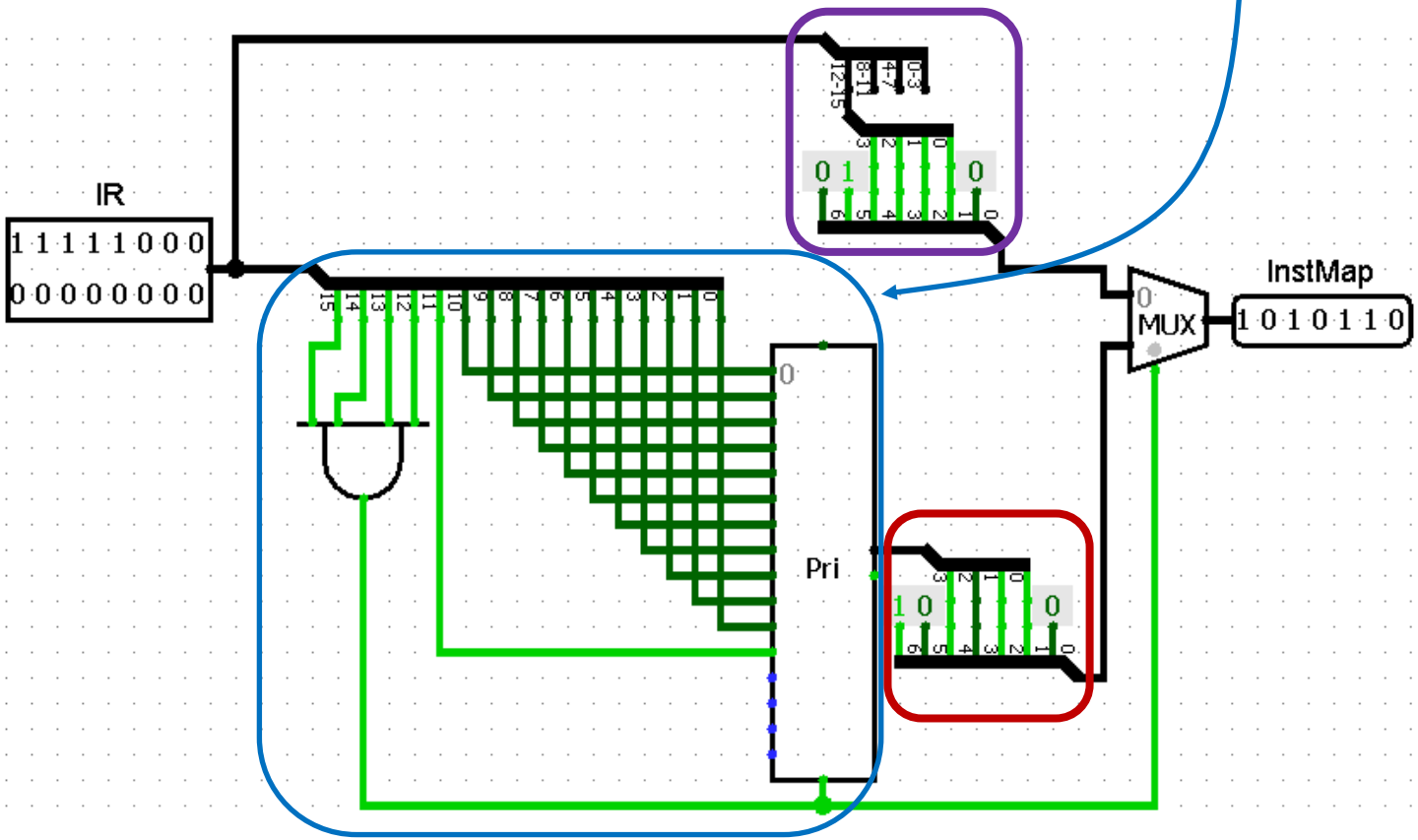



# ALU와 carry 제어 모듈 연결



# Register reference instruction encoding

- Memory reference instruction: 인코딩된 4비트 사용
  - 20h ~ 3Eh 공간에 매핑
- Register reference instruction encoding: 4비트로 인코딩
  - 40h ~ 5Eh 공간에 매핑.



# InstMap

## Memory reference instruction

| | Instruction | Hex Code | MAP Address |
|---|---|---|---|
| 1 | AND | 0xxx | 20 |
| 2 | ADD | 1xxx | 22 |
| 3 | LDA | 2xxx | 24 |
| 4 | STA | 3xxx | 26 |
| 5 | BUN | 4xxx | 28 |
| 6 | BSA | 5xxx | 2A |
| 7 | ISZ | 6xxx | 2C |
| 8 | | | |
| 9 | | | |
| 10 | SUB | 9xxx | 32 |
| 11 | XCH | Axxx | 34 |
| 12 | | | |
| 13 | BPA | Cxxx | 38 |

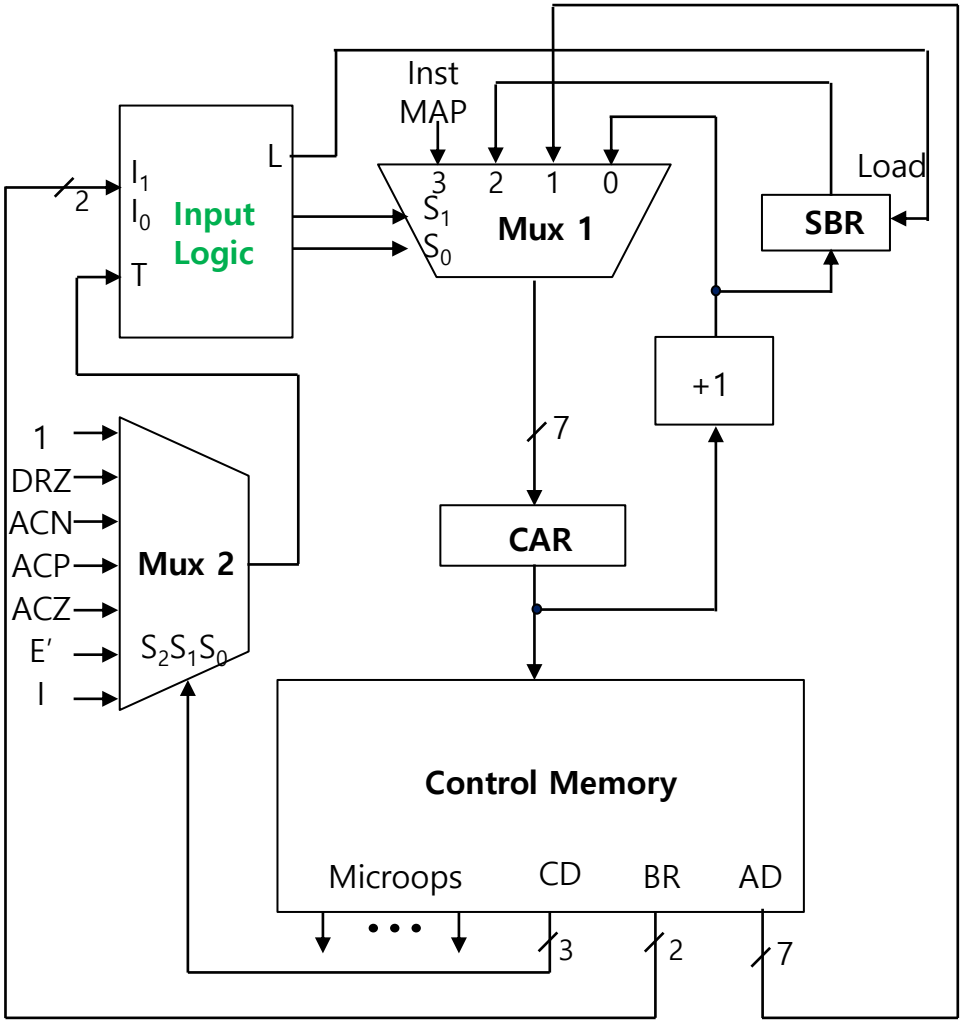
## Register reference instruction

| | Instruction | Hex Code | MAP Address |
|---|---|---|---|
| 1 | CLA | F400 | 40 |
| 2 | CLE | F200 | 42 |
| 3 | CMA | F100 | 44 |
| 4 | CME | F080 | 46 |
| 5 | CIR | F040 | 48 |
| 6 | CIL | F020 | 4A |
| 7 | INC | F010 | 4C |
| 8 | SPA | F008 | 4E |
| 9 | SNA | F004 | 50 |
| 10 | SZA | F002 | 52 |
| 11 | SZE | F001 | 54 |
| 12 | HLT | F800 | 56 |

# Sequencer and Condition Flag

- HW4와 유사
  - Mux2 부분만 확장

| CD | 신호 이름 | |
|---|---|---|
| 0 | Always | |
| 1 | DRZ | DR == 0 |
| 2 | ACN | AC < 0 |
| 3 | ACP | AC >=0 |
| 4 | ACZ | AC == 0 |
| 5 | EZ | Carry == 0 |
| 6 | IR(15) | I bit |



# 제어신호 배치와 Control Memory 설계

- control memory address bits : 7 비트
- Hw2 에서 찾아낸 제어신호들을 Excel sheet에 배치한다.
- Hw2 에서 구한 각 명령어의 Microoperation 에 해당하는 제어신호들을 엑셀 sheet에 입력하여 각 마이크로 명령어에 해당하는 16 진수 microcode를 완성한다. (Hw4 참조)
- Excel sheet에 배치된 제어신호 순서대로 Control Memory Output 에 Tunnel 을 연결하여 회로를 완성한다.

# 테스트 프로그램 1

```
        ORG 0
        LDA     A       / Load 1st Data
        ADD     B       / Add 2nd Data
        STA     C       / Store result
        HLT
A,      DEC 83
B,      DEC -23
C,      DEC 0
        END
```

# 테스트 프로그램 2

```
        ORG     0
        LDA     X
        BSA     MULT4           / CALL
        STA     Y
        HLT

MULT4, HEX      0
        CLE
        CIL
        CIL
        BUN     MULT4 I         / return

X,      HEX     1011
Y,      HEX     0               / result
        END
```

# 테스트 프로그램 3

```
        ORG     0
        LDA     NUMB
        CMA
        INC
        STA     CNT     / put -NUMB to CNT
        LDA     DATP
        STA     PTR
        CLA             / so we can do it again
LOOP,   ADD     PTR I   / add next value to AC
        ISZ     PTR     / point at next element
        ISZ     CNT     / increment counter
        BUN     LOOP    / loop if cnt != 0
        STA     SUM     / done, sum <- AC
        HLT

NUMB,   DEC     8       / number to add up
DATP,   HEX     20      / start of array
SUM,    HEX     0       / result
PTR,    HEX     0       / current location
CNT,    HEX     0       / countup counter

        ORG     20      / data here
        DEC     1       / [0], N =  1
        DEC     2       / [1], N =  2
        DEC     4       / [2], N =  3
        DEC     8       / [3], N =  4
        DEC     16      / [4], N =  5
        DEC     32      / [5], N =  6
        DEC     64      / [6], N =  7
        DEC     128     / [7], N =  8
        END
```

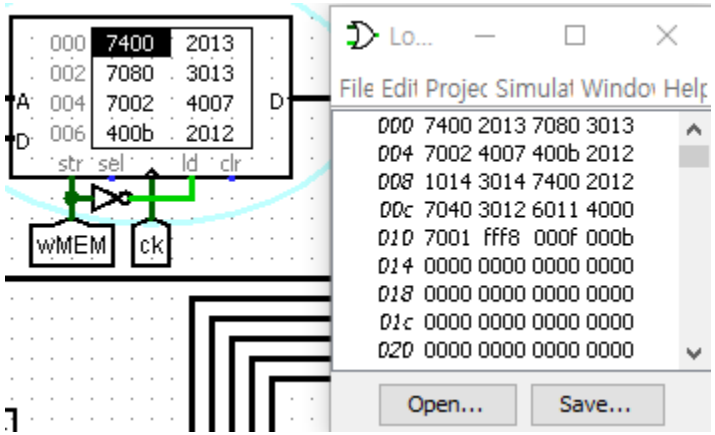
# 테스트 프로그램 4

```
        ORG     0
        ISZ     DATCNT
STRLP,  LDA     DATCNT
        STA     CNT
        BSA     SUBA    // call subroutine
        ISZ     DATBP
        ISZ     DATCNT
        BUN     STRLP
        HLT

SUBA,   HEX     0       // return address
        LDA     DATBP
        STA     PTR
LOOP,   LDA     DATBP I // load 1st data
        ISZ     PTR     // check next data
        SUB     PTR I   // 1ST DATA - DATA
        BPA     NEXT    // next if MAX > data
        LDA     DATBP I
        XCH     PTR I   // exchange A <-> data
        XCH     DATBP I // exchange A <-> 1st data

NEXT,   ISZ     CNT     // increment counter
        BUN     LOOP    // loop if cnt != 0
        BUN     SUBA I  // return, indirect
```

```
PTR,    HEX 0           // index
CNT,    HEX 0           // counter

DATBP,  HEX 20          // start of data
DATCNT, DEC -4          // - num data

        ORG     20
DATA,   HEX     0005
        HEX     0002
        HEX     0017
        HEX     0009

        END
```

# Copy and paste of Test Program

- Ex-Mano-Simulator 에서 프로그램이 실행되는 것을 확인한다.
- RAM 내용을 'Export' 단추를 이용해서 내보내어 Clipboard에 copy한다. 또는, 일부를 dragging 하여 copy 한다.
- Logisim 회로에서 RAM의 edit-contents 메뉴를 선택하여 원하는 주소에 '붙여넣기'를 하면 된다.
- 실행하여 결과를 확인한다.



# 제출물

- logisim 회로 파일 (파일이름: hw5-학번.crc)
  - 마이크로 프로그램이 포함되어 실행 가능한 최종 logisim 파일
  - 메인 회로 좌측, 위쪽 빈공간에 문서 이름과 저자 정보

    Hw5 Mano CPU 구현

    학번: 2011111

    이름: 홍길동

- Excel 파일
  - 마이크로 프로그램을 제작한 Excel 파일 (파일이름: hw5-학번.xlsx)
  - 마이크로 코드 우측에 마이크로 오퍼레이션 나타낼 것.

- 마감일
  - 12월 6일(화) 23시59분