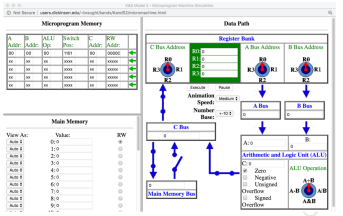
09 – MACHINE LANGUAGE

COMP256 – COMPUTING ABSTRACTIONS

DICKINSON COLLEGE

RAISING THE LEVEL OF ABSTRACTION

- Micro-program instructions require thinking about the internal machine details and how they need to be configured to carry out a desired operation.
  - Complex distraction from the task at hand
  - Possible to specify meaningless micro-instructions
- A higher level of abstraction (e.g. machine language) allows programmers to focus on the desired operations (e.g. move data, add, etc…) without worrying about details such as switch positions.
  - Increases efficiency
  - Reduces mistakes

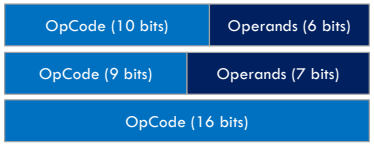


MACHINE LANGUAGE

- A *machine language* is created by defining patterns of bits to correspond to common operations (e.g. moving data, addition, etc).
  - Machine language is translated to micro-instructions by the control unit.
  - The control unit:
    - *fetches* a machine language instruction from main memory,
    - *decodes* the instruction,
    - and configures the machine (via a micro-instruction) to *execute* the instruction.
- Each processor has its own unique machine language.

MACHINE LANGUAGE INSTRUCTIONS

- Machine language instructions have two parts:
  - The *OpCode* (operation code) specifies the operation to be performed.
  - The *Operands* (arguments) specify the data to be used in the operation.
- The *format* of a machine language instruction specifies how its bits encode the opcode and its operands.
  - For Example three of the instruction formats for the Knob & Switch computer are:



KNOB & SWITCH MACHINE LANGUAGE

- Data Movement Instructions

1000 0001 0	RegC(2) address (5)	RegC = MM[address]
1000 0010 0	RegC(2) address (5)	MM[address] = RegC
1001 0001 0000	RegC (2) RegA (2)	RegC = RegA

KNOB & SWITCH MACHINE LANGUAGE

- Arithmetic and Logic Instructions:

1010 0001 00	RegC (2) RegA(2) RegB(2)	RegC = RegA + RegB
1010 0010 00	RegC (2) RegA(2) RegB(2)	RegC = RegA - RegB
1010 0011 00	RegC (2) RegA(2) RegB(2)	RegC = RegA & RegB
1010 0100 00	RegC (2) RegA(2) RegB(2)	RegC = RegA | RegB

KNOB & SWITCH MACHINE LANGUAGE

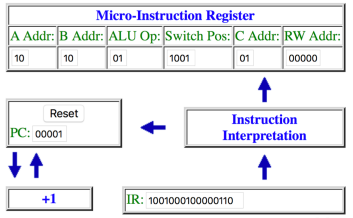
- One other Instruction:

1111 1111 1111 1111	Halt

- The Halt instruction indicates the end of the program.
- All Knob & Switch programs should end in with a Halt instruction.

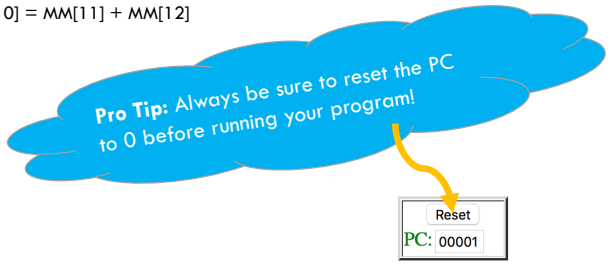
THE CONTROL UNIT

- The *control unit* directs the fetch, decode execute cycle:
  - Fetch:
    - The memory address in the *Program Counter* (PC) is used to fetch from memory the next program instruction to process.
      - Starts at 00000_2
    - The instruction to be processed is placed into *Instruction Register* (IR).
  - Decode:
    - The *instruction interpretation* circuitry decodes the machine language instruction and translates it into a micro-instruction.
  - Execute:
    - The machine executes the instruction based upon the bits in the micro-instruction register.
    - Program counter is updated to address of next instruction.



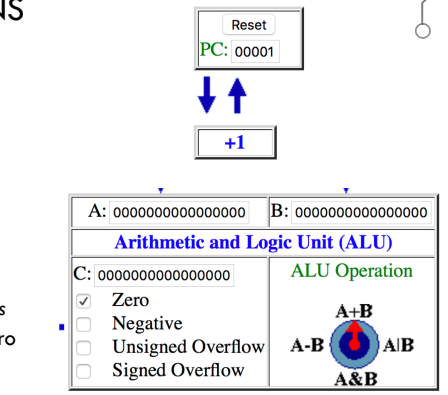
EXAMPLE

- Give a machine language program (i.e. a sequence of machine language instructions) that accomplish the following operation and then stop:
  - MM[10] = MM[11] + MM[12]



BRANCHING INSTRUCTIONS

- *Branching instructions* provide the mechanism for programs to make decisions (e.g. if/else, loops).
  - Branching instructions change the address from which the next instruction will be fetched by changing the Program Counter (PC)
- Branches are based upon the *ALU Flags* from the most recent operation (e.g. zero result, or negative result).



KNOB & SWITCH MACHINE LANGUAGE

- Branching Instructions:

0000 0001 000	address (5)
0000 0010 000	address (5)
0000 0011 000	address (5)

Branch: PC = address

Branch on Zero:
If ALU Zero Flag is set then
PC = address
else
PC = PC + 1

Branch on Negative:
If ALU Negative Flag is set then
PC = address
else
PC = PC + 1

ACTIVITIES

- Write machine language programs that do the following:
  - If MM[20] < 0 then MM[20] = 0
    - Note that this is similar to the high level language:
      - `If x < 0 then x = 0`
  - If MM[20] == 0 then MM[21] = MM[22], else MM[21] = MM[23]
    - Note that this is similar to the high level language:
      - `if x == 0, y = a else y = b`

ACTIVITY SOLUTION

- If MM[20] < 0 then MM[20] = 0

MM	Instruction	Effect
0:	1000 0001 0 00 10010	R0 = MM[20]
1:	1010 0100 00 00 00 00	R0 = R0 | R0 (Pass R0 through ALU)
2:	0000 0011 000 00100	Branch to 4 if negative
3:	1111 1111 1111 1111	Halt
4:	1010 0010 00 00 00 00	R0 = R0 – R0 (make zero)
5:	1000 0010 0 00 10010	MM[20] = R0
6:	0000 0001 000 00011	Branch to 3 (to Halt)