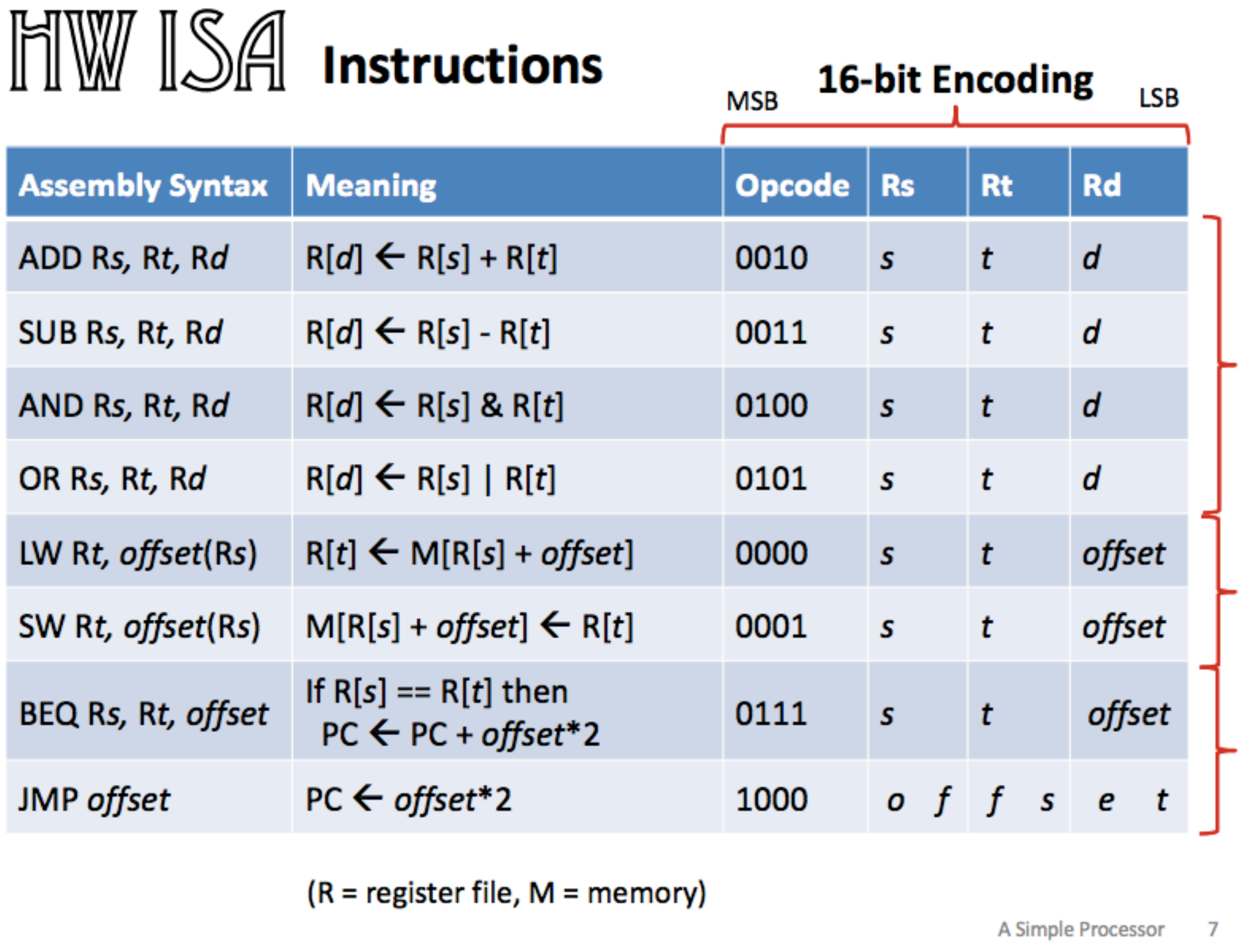
Laboratory 6
Processor Datapath

Description of HW Instruction Set Architecture

- **16 bit data bus**
- **8 bit address bus**
- **16 registers**
  R0 = 0 (constant)
  R1 =1 (constant)
  R2-R15 general purpose



HW ISA **Instructions**

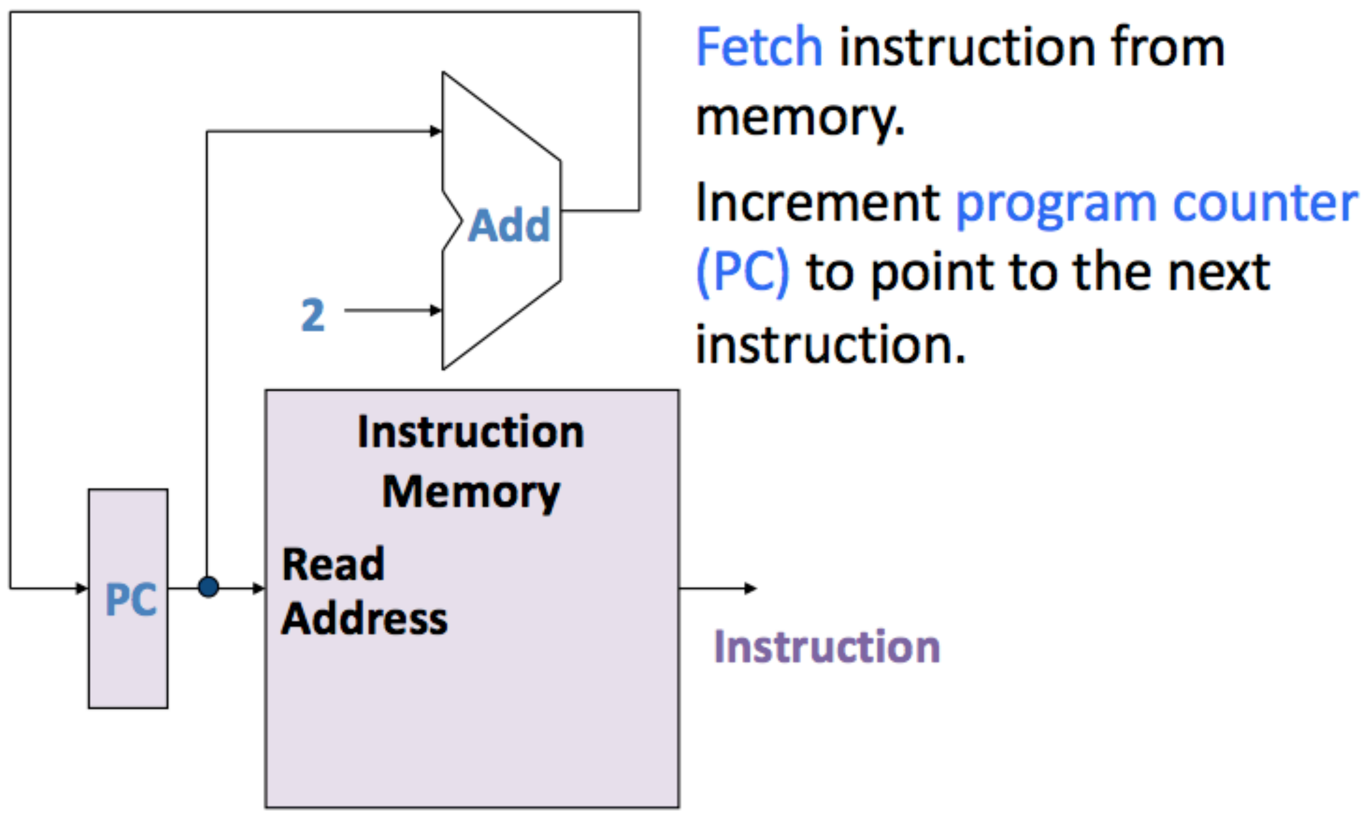
MSB **16-bit Encoding** LSB

Assembly Syntax	Meaning	Opcode	Rs	Rt	Rd
ADD Rs, Rt, R*d*	R[*d*] ← R[*s*] + R[*t*]	0010	*s*	*t*	*d*
SUB Rs, Rt, R*d*	R[*d*] ← R[*s*] - R[*t*]	0011	*s*	*t*	*d*
AND Rs, Rt, R*d*	R[*d*] ← R[*s*] & R[*t*]	0100	*s*	*t*	*d*
OR Rs, Rt, R*d*	R[*d*] ← R[*s*] | R[*t*]	0101	*s*	*t*	*d*
LW Rt, ***offset***(Rs)	R[*t*] ← M[R[*s*] + ***offset***]	0000	*s*	*t*	***offset***
SW Rt, ***offset***(Rs)	M[R[*s*] + ***offset***] ← R[*t*]	0001	*s*	*t*	***offset***
BEQ Rs, Rt, ***offset***	If R[*s*] == R[*t*] then PC ← PC + ***offset****2	0111	*s*	*t*	***offset***
JMP ***offset***	PC ← ***offset****2	1000	*o f*	*f s*	*e t*

(R = register file, M = memory)

Fetch Instruction from Memory

- **PC** register holds address of currently executing instruction
- Programs are assumed to start at address 0
- **PC** initialized to 0 by a reset to begin execution
- Next instruction located at current **PC + 2**



Fetch instruction from memory.
Increment program counter (PC) to point to the next instruction.
Add
2
Instruction Memory
Read Address
PC
Instruction

Branch Address

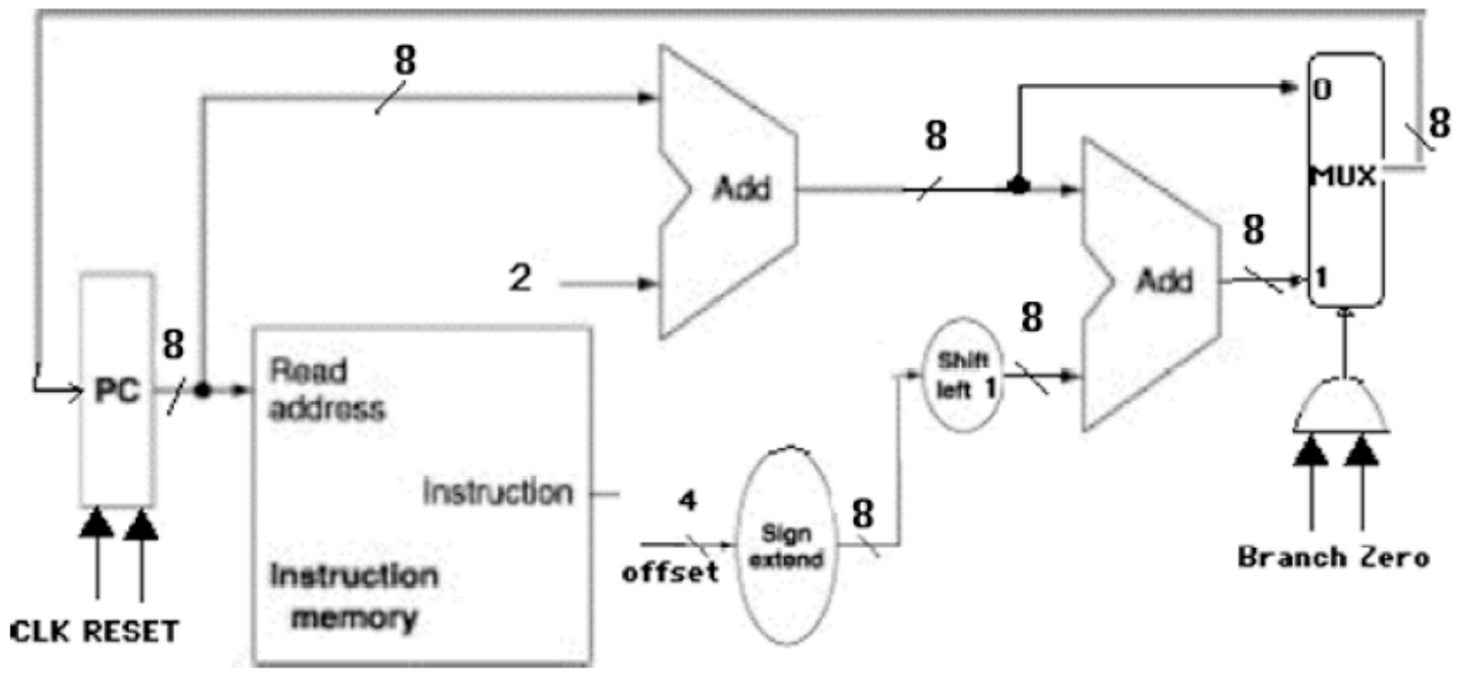
Programs do not always execute in sequential order.

When the **BEQ** instruction is executed, the next instruction to be executed is either:

PC = PC + 2

or

PC = PC + 2 + (2*offset)



BEQ Rs,Rt,offset

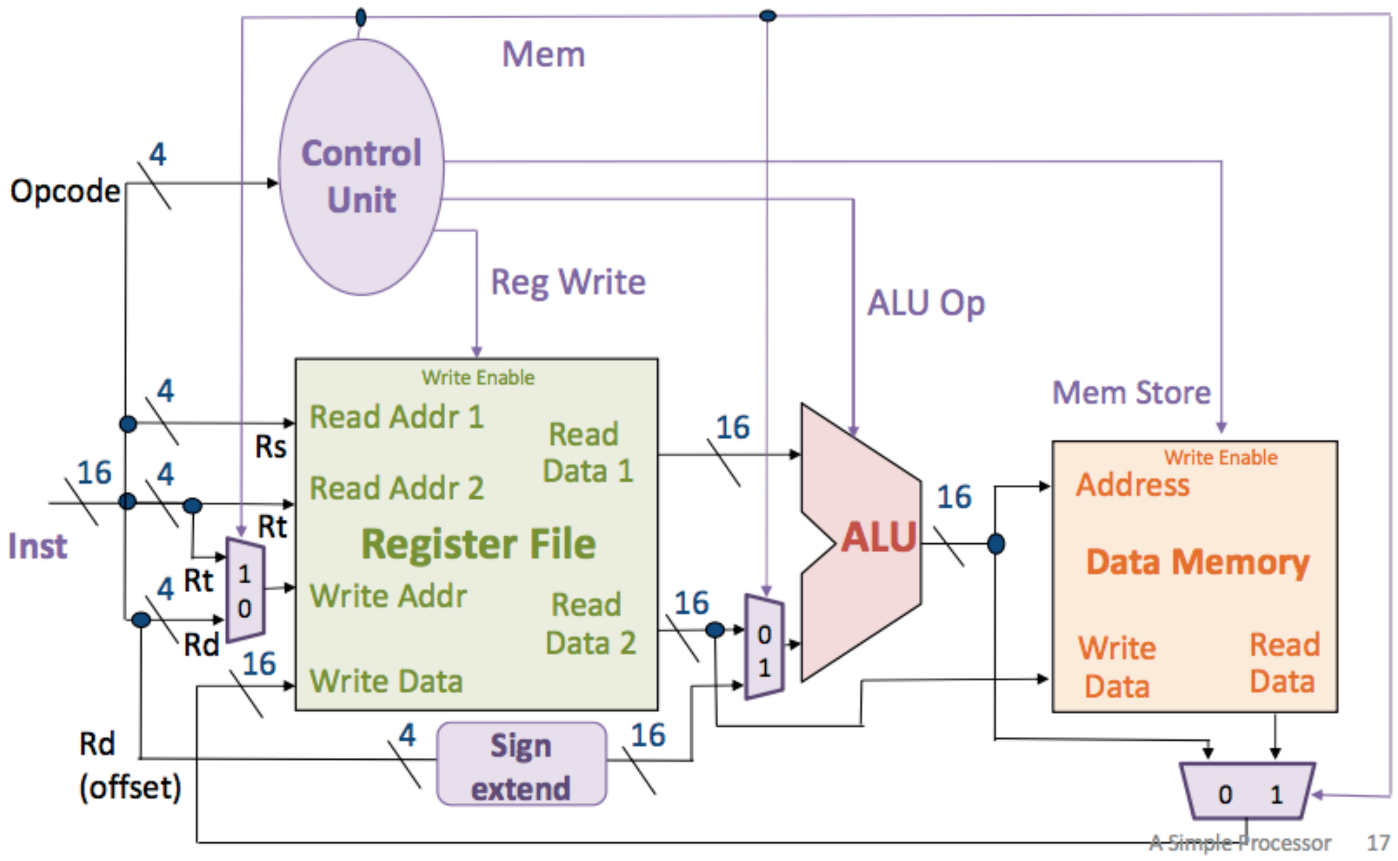
- The **offset** = number of instructions away from the next value of the PC to branch to, so must be multiplied by 2.
- Since **offset** is 4 bits, it must be sign-extended to 8 bits to be added to the PC.

A **MUX** selects the next value of the PC. The value of the **Branch** and **Zero** bits are used to determine which is used:

- The **Branch** control line = 1 if a BEQ instruction is being executed.

- The **Zero** bit from the ALU is used to check whether Rs = Rt: it is 1 if Rs – Rt = 0 (meaning they're equal). If **Branch** = 1 and **Zero** = 1, then the next value of the PC will be the branch address ; otherwise, it will simply be **PC + 2**

Datapath



R-type instructions ADD,SUB,AND,OR   have format: **opcode Rs Rt Rd**

- read **Rs** and **Rt** from register file
- perform an ALU operation on the contents of the registers
- write the result to register **Rd** in register file

Memory Access instructions LW,SW   have format: **opcode Rt offset(Rs)**

- **offset** is in the position in the instruction where **Rd** is for R-type instructions
- memory address = **Rs** + sign-extended 4-bit offset
- if **SW**, the value to be stored to memory is from **Rt**
- if **LW**, **Rt** is loaded with the value read from memory

Register written to in regfile is specified by **Write Addr**, which is either **Rd** or **Rt** (chosen by MUX which is controlled by **MemLoad**)

ALU operates on **Rs** and **Rt**, or on **Rs** and the sign-extended **offset**

- Input A of the ALU is always **Rs**
- Input B of the ALU is **Rt or** the **offset** (chosen by a MUX controlled by **MemLoad)**

Control Unit

ALU can perform 4 possible operations:

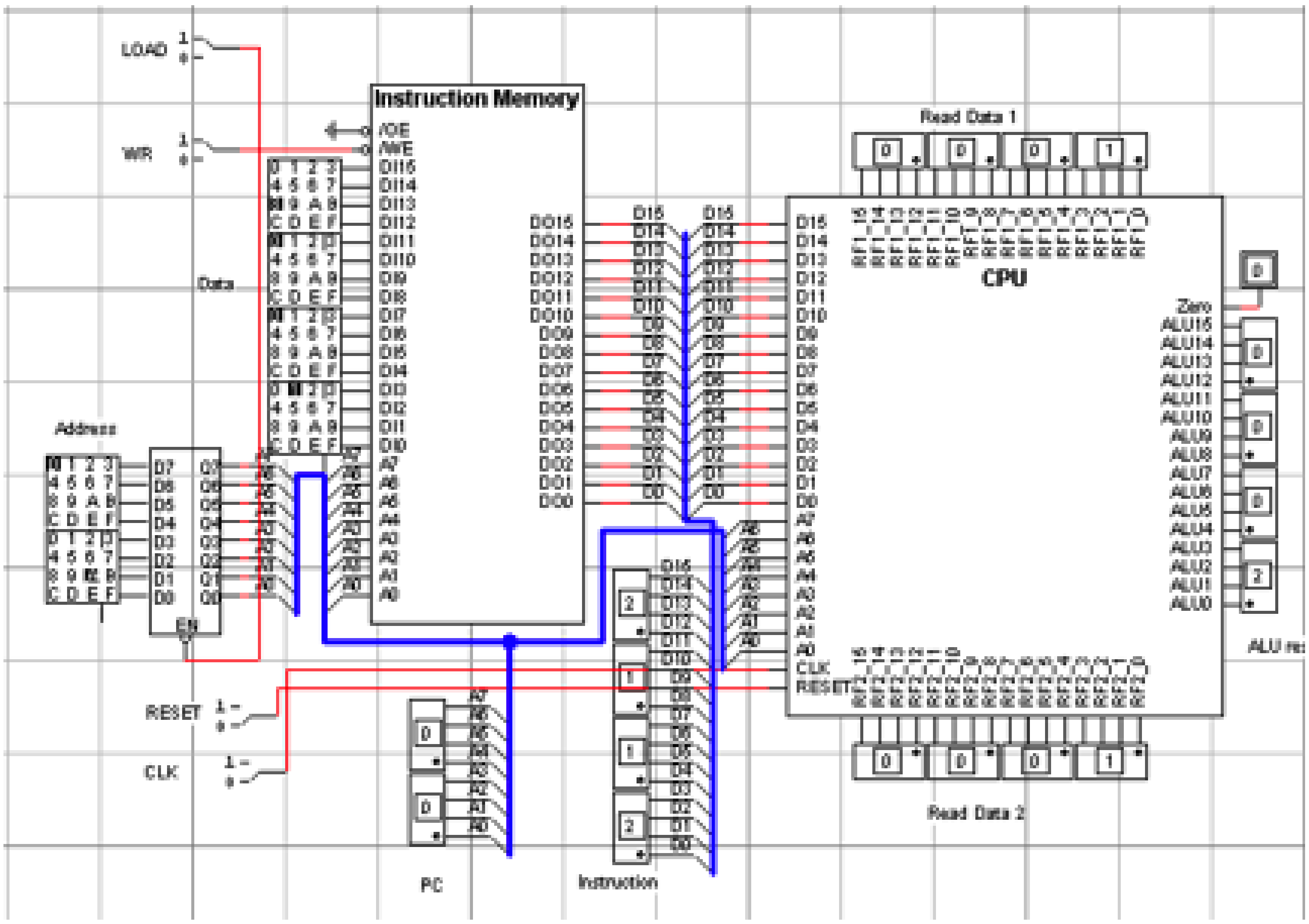
ALUop	ALU function
0000	a AND b
0001	a OR b
0010	a + b (add)
0110	a - b

The Control Unit select the proper ALU operation for each instruction, along with the proper control signals, based on the opcode:

Instruction	Opcode	ALUop	RegWrite	MemLoad	MemStore	Branch	Jump
LW	0000	0010	1	1	0	0	0
SW	0001	0010	0	1	1	0	0
ADD	0010	0010	1	0	0	0	0
SUB	0011	0110	1	0	0	0	0
AND	0100	0000	1	0	0	0	0
OR	0101	0001	1	0	0	0	0
BEQ	0111	0110	0	0	0	1	0
JMP	1000	don’t care	0	0	0	0	1

Can use decoders or simple logic to produce these signals.

Full Implementation



Procedure to Load/Execute a New Program

1. Disconnect the address bus of the Instruction Memory from the CPU
2. Set **LOAD** = 0

3. Set **address** and **data** switches for instruction
4. Set **WR** = 0, then back to 1
5. Repeat steps 3 and 4 until all instructions are loaded to memory

6. Set **LOAD** = 1
7. Reconnect address bus to CPU

8. Set **Reset** = 1, then back to 0
9. Set **CLK** = 1, then back to 0, for each instruction.

```
Address              Instruction  _    Rs  Rt  Rd/offset   Purpose
0:                          ADD        R1 R1 R2            ; R2 = 2
2:                          ADD        R2 R2 R3            ; R3 = 4
4:                          ADD        R3 R3 R3            ; R3 = 8
6:                          SW         R0 R3  0            ; data address 0: 8
8:                          SW         R0 R2 2             ; data address 2: 2
A:(LOOP)                    LW         R0 R5 0             ; R5 <- 8 from address 0
C:                          LW         R0 R4 2             ; R4 <- 2 from address 2
E:                          SUB        R5 R4 R5            ;R5<-  8 – 2 = 6
10:                         SW         R0 R5 4             ;data address 4:  6
12:                         LW         R0 R15 4            ;R15 <- contents of address 4
14:                         OR         R15 R15 R15         ; displays  R15 at ALU result
16:                         BEQ        R2 R15 LOOP         ; repeat the starting at label loop
18:                         J           END                ; jump to end of program
…
1E:  (END):
```