Multi-cycle CPU (Also refer to Professor Roumani's slides for on this.)

The Critiques of Single-cycle:

- Positive ☺: CPI = 1
- Positive ☺: Simple
- Negative ☹: Waste of hardware resources:
  e.g. A complete ALU is used just to add 4 to PC.
- Negative ☹: Caters for slowest: Clock rate must go with the slowest instruction:
  Currently these are the approximate latencies for the fastest CPUs:

Instruction	IM	RF	ALU	DM	WB	Total
R-Type	2	1	2	0	1	6
lw / sw	2	1	2	2	1 / 0	8 / 7
branch	2	1	2	0	0	5
jump	2	0	0	0	0	2

In this example the slowest latency is 8 ns, so the clock rate will be:

$$Clock\ rate = \frac{1}{(8\ ns)} = \frac{10^9}{8} Hz = 125\ MHz$$

But we know today processors are a many times faster than 125 MHz.

The multi-cycle design...

- Has no hardware redundancy ☺
- Can not store data in wires (i.e. needs more registers) ☹
  In addition to PC and RF from single-cycle, we also need A, B, MDR, IR and ALUout (MAR)
  These registers keep the results from previous cycles. For example in the diagram you will see a direct bus out of ALU that contains the results from the current cycle and one from ALUout that contains the results from the previous cycle.
- BEQ computation is in serial (unlike single-cycle that was in parallel)
  Yet, it is done in only 3 cycles – one of the shortest instructions.
- Cycle is based on largest latency (i.e. slowest device), not longest instruction
  In one cycle, either IF, DM, RF or ALU

See the multi-cycle data path in Professor Roumani's notes or in the textbook. (fig 5.33/5.42)

Note: In a digital circuits, we can take two copies of the same wire, but can never join two wires. You can see this on the data path figure.

There are two types of controls in the multi-cycle data path:

- State-Element Controls: Controls for the parts of the circuit that hold data (PC, RF, etc.)
  In the multi-cycle data path we always AND the clock signal with these.
  Whenever the state of these controls is not mentioned they are zero.
- Multiplexers and ALU Control
  Whenever the state of these controls is not mentioned, we don't care about that value since we don't care about the data that crosses them.

The cycles...

- **Cycle #0** does two jobs no matter what the instruction is:
  - Instruction Fetch (IorD = 0, MemRead = 1, IRWrite = 1)
  - PC++ (ALUsrcA = 0, ALUsrcB = 01, ALUop = 00, PCSource = 00, PCWrite = 1)
  - In case of BEQ Branch Destination is also computed in this cycle.
- **Cycle #1** is the instruction decode no matter what the instruction is:
  - Look up and Decode (ALUsrcA = 0, ALUsrcB = 11, ALUop = 00)
- **Cycle #2 and on** depend of the instruction
  - J & BEQ are 3 cycles each
  - SW and R-type instructions are 4 cycles
  - LW being the longest is 5 cycles

See the Control diagram in Professor Roumani's notes or in the textbook.

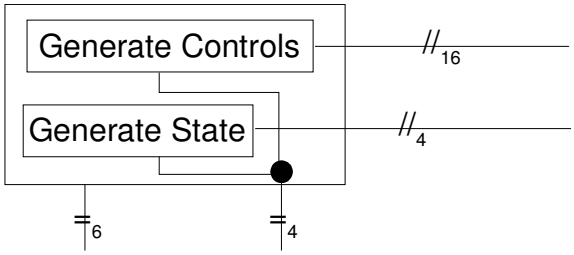
Functional Specifications

- Truth Tables – Tell you the output for a given input
- FSM (Finite State Machine) - Tell you the output for a given input *at a given time/state*

Implementation of Control for Multi-cycle

The numbers in the control diagram are state numbers (from 0-9)

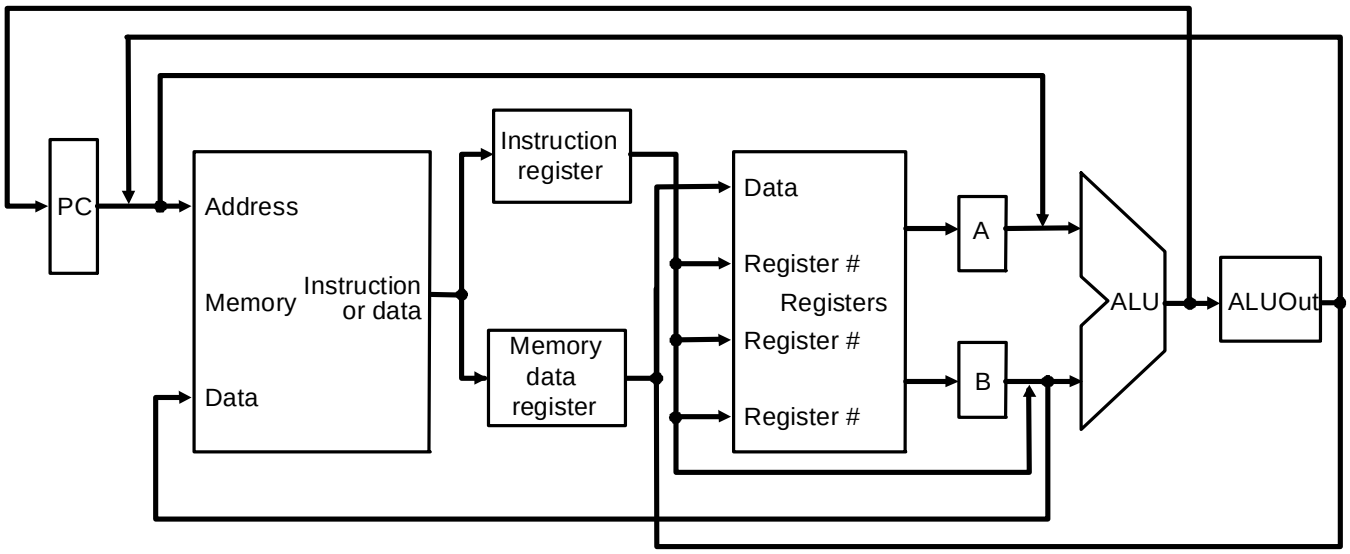
The OpCode is only needed to generate these state numbers. Once we have the states we know what the control signals are supposed to be.

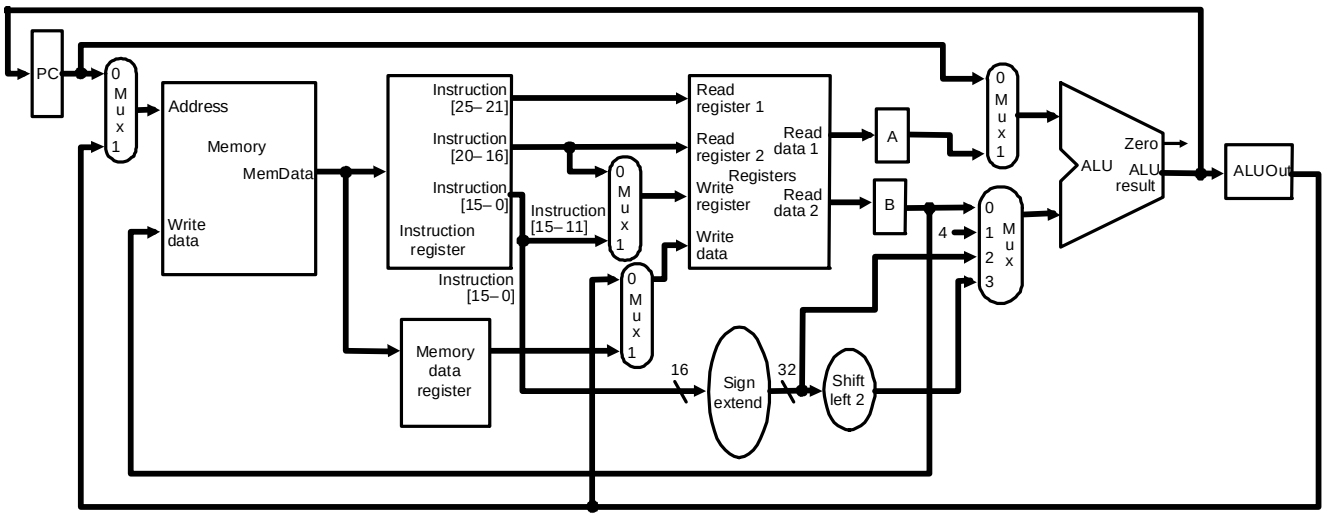


This way instead of a truth table with 10 inputs and 20 outputs we have two tables: one with 4 inputs and 16 outputs and another with 10 inputs and 4 outputs.

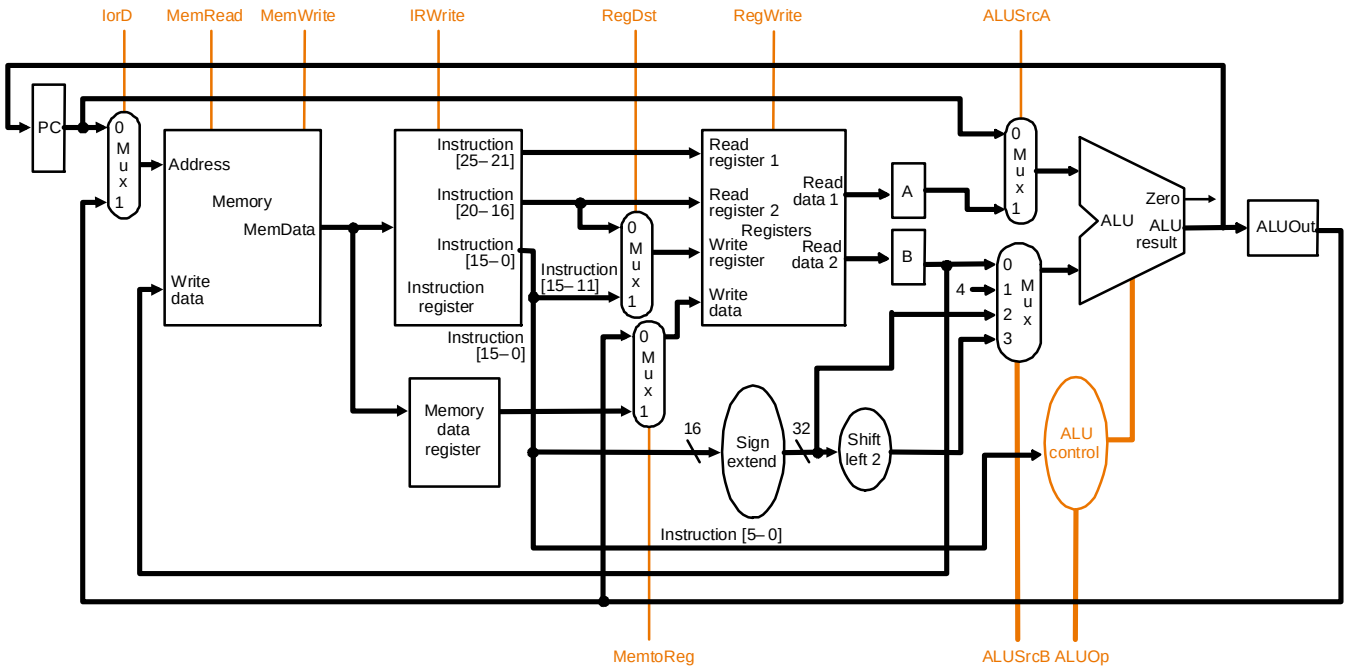
Of course, we can easily program a PLA with those truth tables.

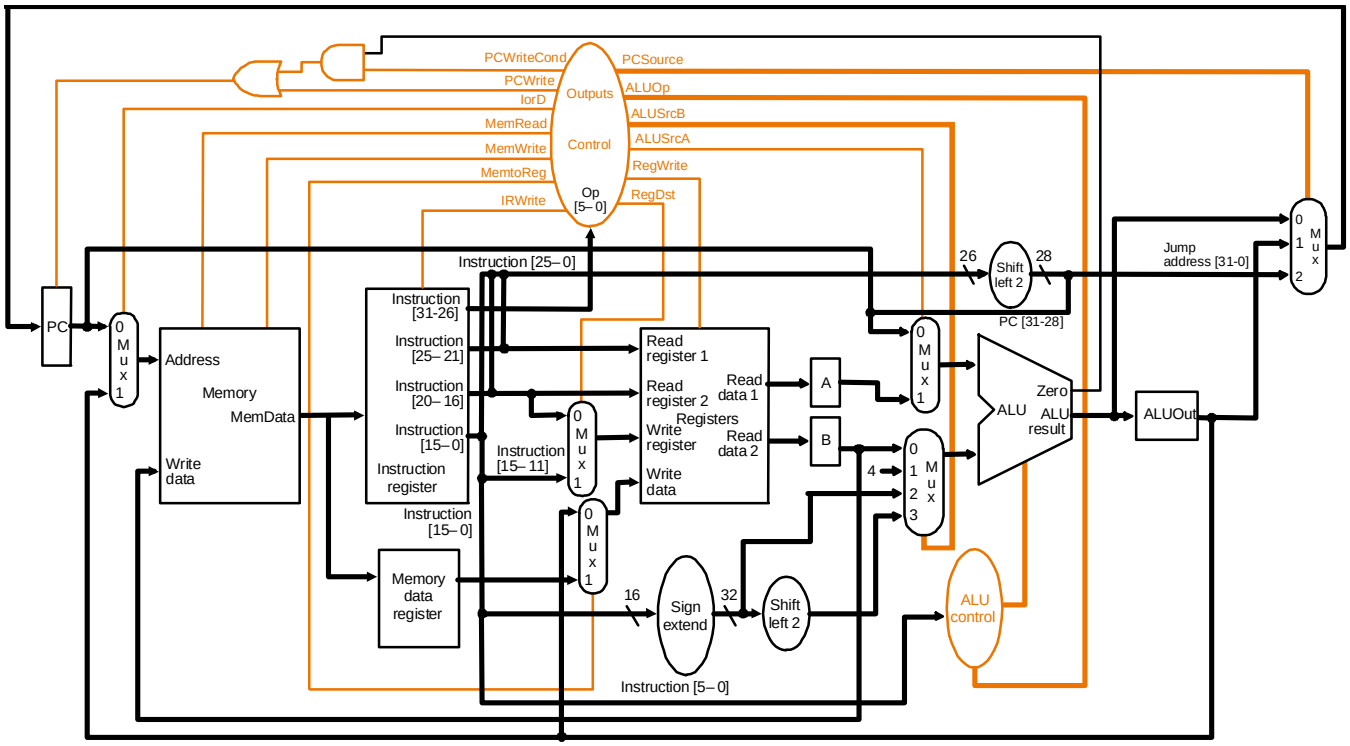
DATAPATH-I


PC
Address
Memory
Instruction or data
Data
Instruction register
Memory data register
Data
Register #
Registers
Register #
Register #
A
B
ALU
ALUOut

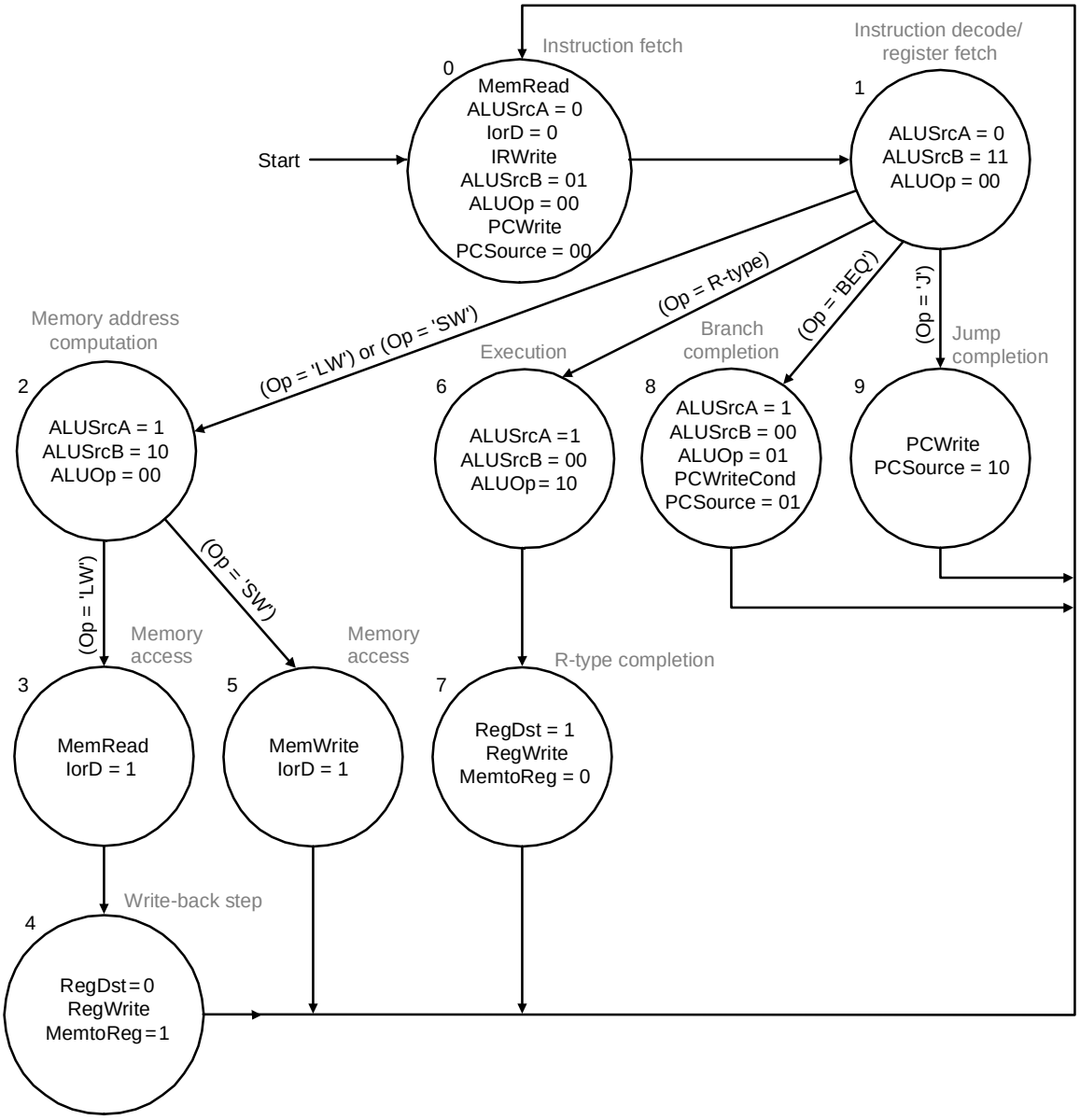

PC
0 M u x 1
Address
Memory
MemData
Write data
Instruction [25– 21]
Instruction [20– 16]
Instruction [15– 0]
Instruction register
Instruction [15– 11]
Instruction [15– 0]
Memory data register
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
16
Sign extend
32
Shift left 2
0 1 2 3 M u x
ALU
Zero
ALU result
ALUOut

DATAPATH-II

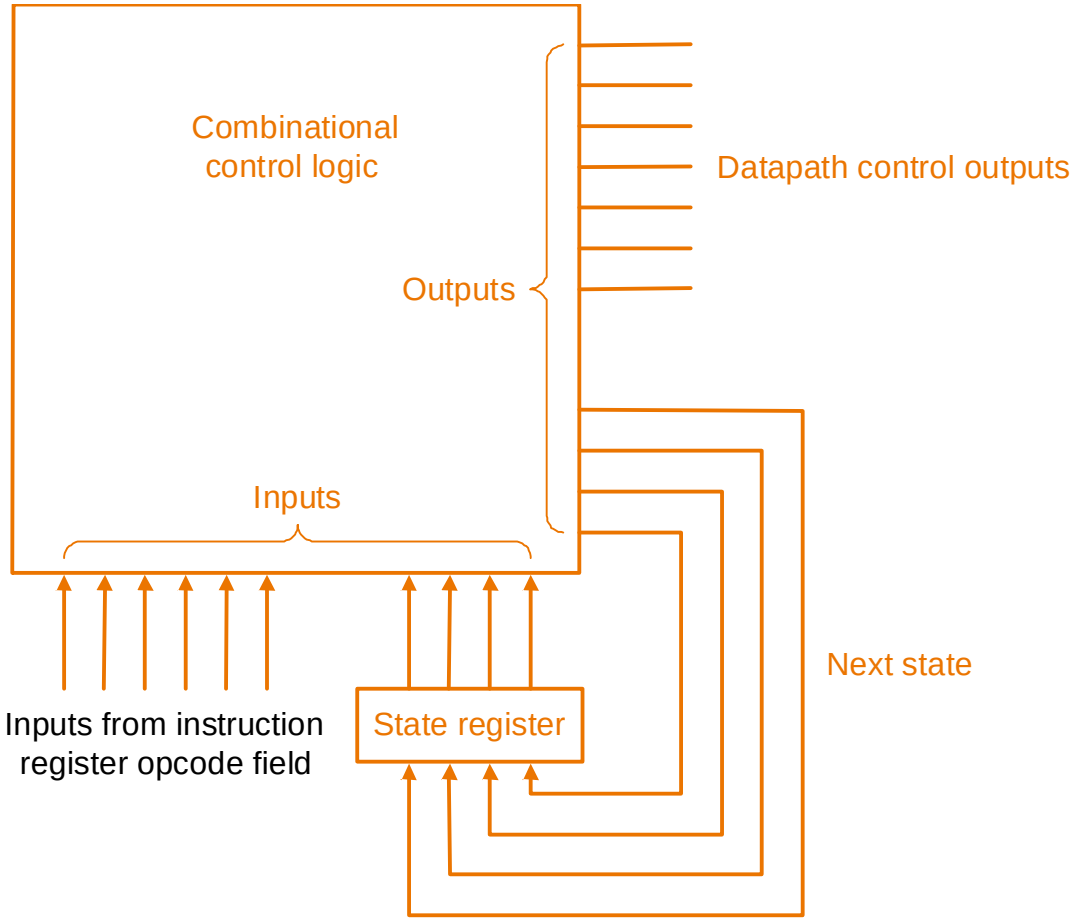

IorD
MemRead
MemWrite
IRWrite
RegDst
RegWrite
ALUSrcA
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[25– 21]
Instruction
[20– 16]
Instruction
[15– 0]
Instruction
register
Instruction
[15– 11]
Instruction
[15– 0]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 1
A
B
4
0
1
2
3
ALU
Zero
ALU
result
ALUOut
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5– 0]
MemtoReg
ALUSrcB
ALUOp


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5– 0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25– 0]
26
Shift
left 2
28
PC [31-28]
Jump
address [31-0]
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[31-26]
Instruction
[25– 21]
Instruction
[20– 16]
Instruction
[15– 0]
Instruction
register
Instruction
[15– 11]
Instruction
[15– 0]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
0
1
2
3
ALU
Zero
ALU
result
ALUOut
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5– 0]

CONTROL FSM


Start
0
Instruction fetch
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
1
Instruction decode/
register fetch
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
2
Memory address
computation
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
Execution
ALUSrcA =1
ALUSrcB = 00
ALUOp= 10
8
Branch
completion
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
Jump
completion
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
3
Memory
access
MemRead
IorD = 1
5
Memory
access
MemWrite
IorD = 1
7
R-type completion
RegDst = 1
RegWrite
MemtoReg = 0
4
Write-back step
RegDst=0
RegWrite
MemtoReg=1

CONTROL IMPLEMENTATION



(Edward) Moore:

Future depends on present state

(George) Meely

Future depends on present state and input.

H/W Implementation