Single Cycle Processor

Chapter 4

Single Cycle Processor

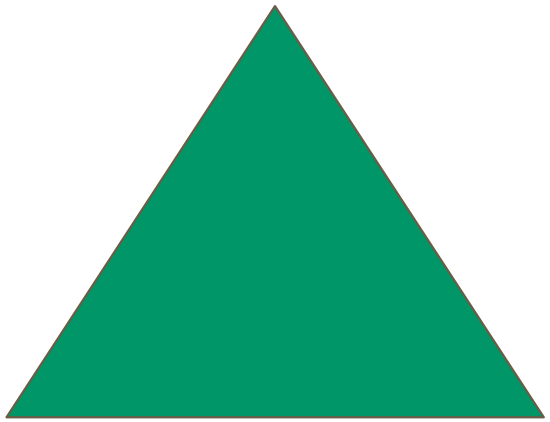
- ★ Design Concept
- ★ Design Steps
- ★ Instruction Set Architecture (nanoLADA)
- ★ Components
  - ○ Register Files
  - ○ Extender
  - ○ Memory
- ★ Data Path
- ★ Control Signal
- ★ Critical Path

Design Concept

- ★ Performance (CPU Time) is a function of
  - Instruction Count
  - Cycle per Instruction
  - Clock Cycle Time

Depends on:
Programs
ISA
Compiler

Instruction Count



Depends on:
Programs
ISA
Compiler?
CPU Organization

Cycle Per Instruction

Depends on:
CPU Organization
VLSI Technology

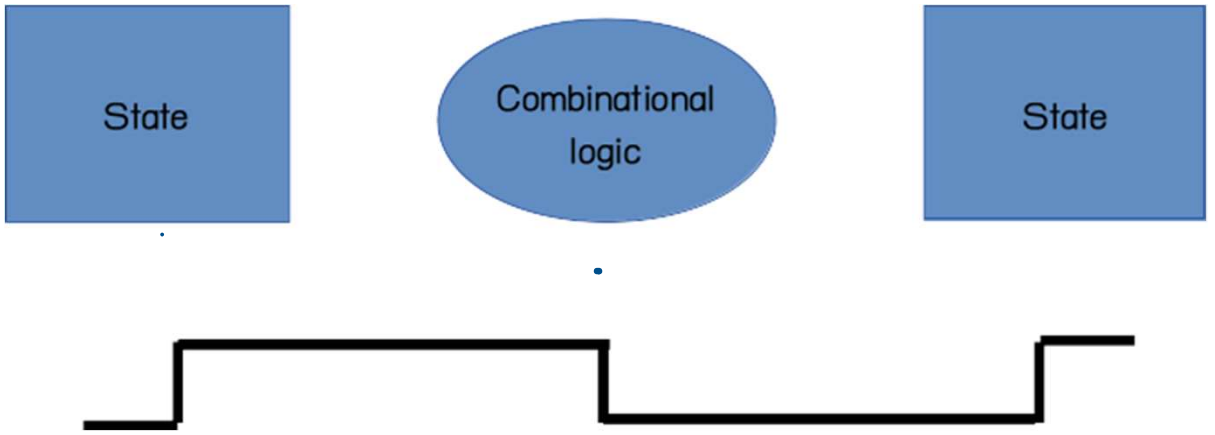
Clock Cycle Time

Design Concept (ctd.)

- ★ Processor Design (DataPath and Control) will determine:
  - ○ Cycle Per Instruction
  - ○ Clock Cycle Time
- ★ We will (try to) design a single-cycle processor
  - ○ 1 cycle per instruction (simple)
  - ○ Very long cycle time (slow)
- ★ In hope that you will understand the internal organization of the processor

Design Concept (ctd.)

- ★ Single-Cycle Processor (CPU = 1)
  - Start an instruction every cycle.
  - Related operation for an instruction is a combinational logic.



Design Steps

1. Analyze the ISA to determine datapath requirements
   a. Definition of each instruction on Register Transfer Language (RTL)
   b. Storage Elements (registers, and more)
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting the requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
5. Assemble the control logic

Compare to to software design?

Instruction Format (nanoLADA)

- ★ Fixed-Length
- ★ 3 formats
- ★ R-Type
  - 6-bit opcode
  - 5-bit RS, 5-bit RT, 5-bit RD
  - 11-bit reserve
- ★ I-Type
  - 6-bit opcode
  - 5-bit RS, 5-bit RT
  - 16-bit immediate
- ★ J-Type
  - 6-bit opcode
  - 26-bit address/displacement

Step 1: Requirements (nanoLADA)

- ★ Instructions are explained with RTL.
- ★ ORI rt, rs, imm ; (I-type) ; Opcode 010000
  - R[rt] ← R[rs] | zero_ext(imm);
  - PC ← PC + 4
- ★ ORUI rt, rs, imm ; (I-type) ; Opcode 010001
  - R[rt] ← R[rs] | zero_pad(imm);
  - PC ← PC + 4
- ★ ADD rd, rs, rt ; (R-type) ; Opcode 000001
  - R[rd] ← R[rs] + R[rt];
  - PC ← PC + 4

Step 1: Requirements (nanoLADA) (ctd.)

- ★ LW rt, imm(rs) ; (I-type) ; Opcode 011000
  - R[rt] ← MEM[R[rs] + sign_ext(imm)];
  - PC ← PC + 4
- ★ SW rt, imm(rs) ; (I-type) ; Opcode 011100
  - MEM[R[rs] + sign_ext(imm)] ← R[rt];
  - PC ← PC + 4

Step 1: Requirements (nanoLADA) (ctd.)

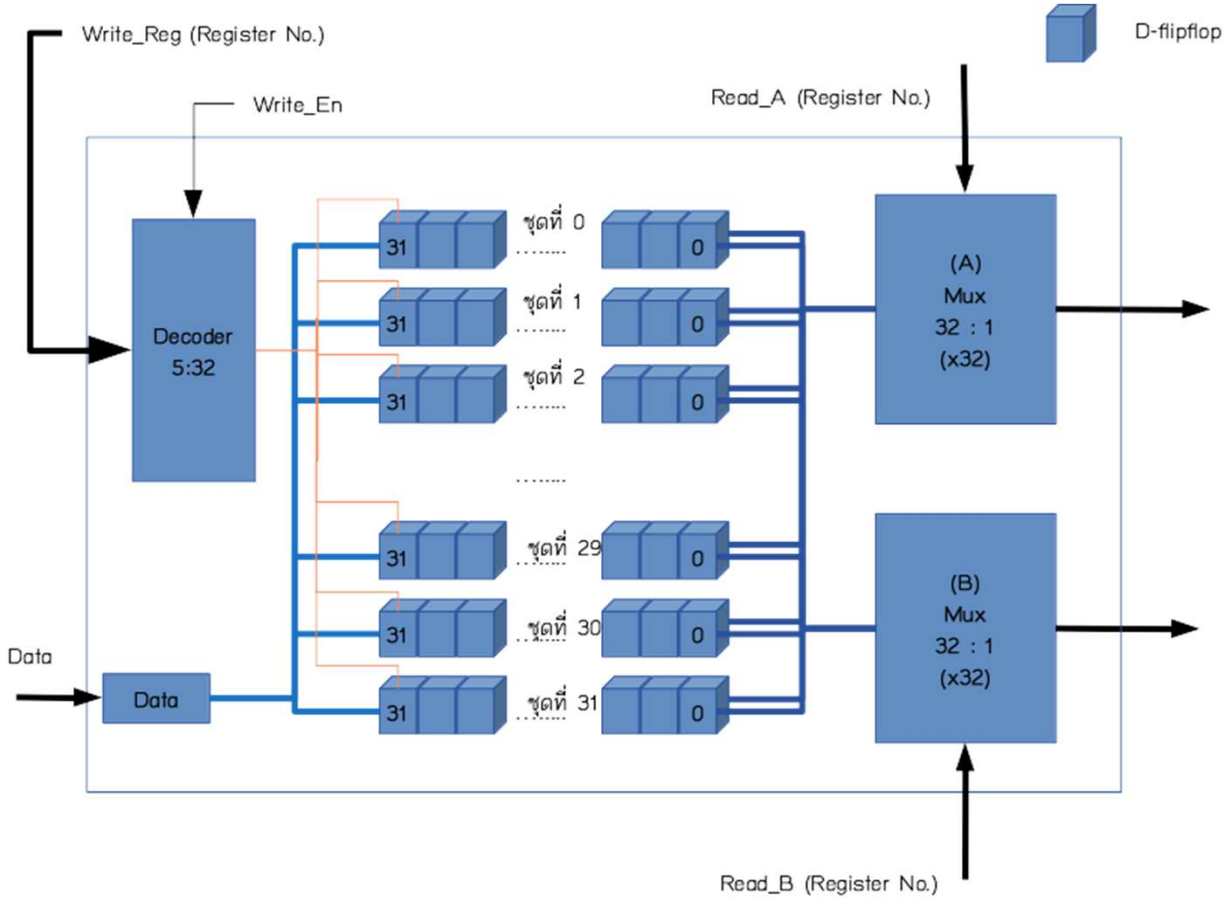
- ★ BEQrs, rt, imm*4 ; (I-type) ; Opcode 100100
  - If (R[rs] == R[rt]) then
  - PC ← PC + 4 + (sign_ext(imm) * 4)
  - else
  - PC ← PC + 4
- ★ JMP addr*4 ; (J-type) ; Opcode 110000
  - PC ← (addr * 4)

Step 2: Components for Datapath

- ★ Register file (bits? size?)
  - ○ Indexed by RS, RT, RD
- ★ Special register
  - ○ PC
- ★ ALU
  - ○ Supports for ADD, OR
- ★ ADDER (for PC)
- ★ EXTENDER

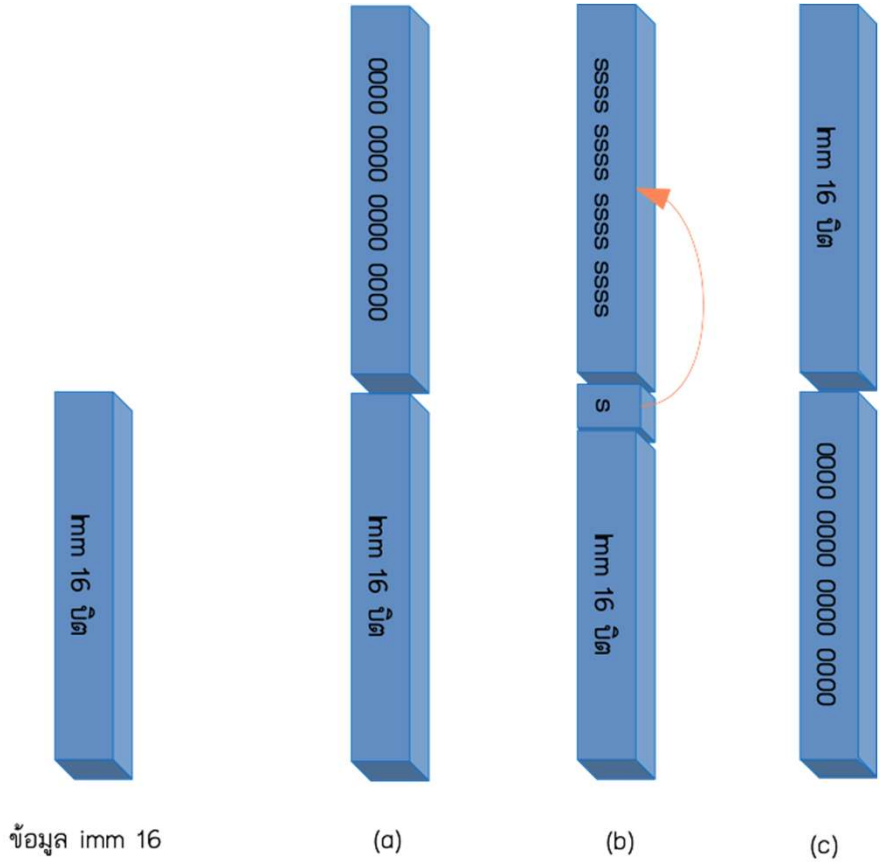
Register File

- ★ 32 (5-bit address) registers
  - Each with 32 bit
- ★ Reading from 2 ports (Read_A, Read_B)
- ★ Writing to one port at Write_Reg
- ★ Can be implemented with 32x32 D-flipflop, 2x muliplexors, and decoder



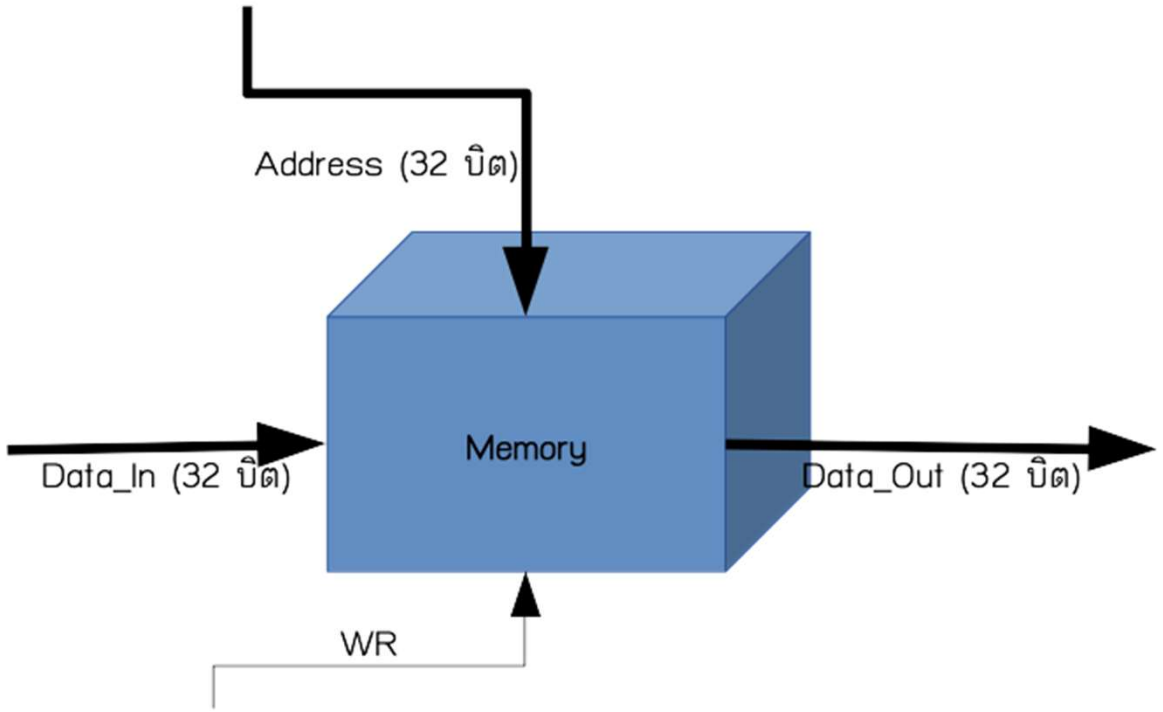
Extender

- ★ Extend 16-bit immediate to 32-bit data
- ★ 3 modes
- ★ Sign Extender (b)
  - Extended with sign bit
- ★ Zero Padding (c)
  - Padded with 16'b0
- ★ Zero Extender (a)
  - Extended with 16'b0



Memory

- ★ A memory
- ★ 32-bit in, out
- ★ 32-bit address
- ★ One port for read
- ★ One port for write



ALU

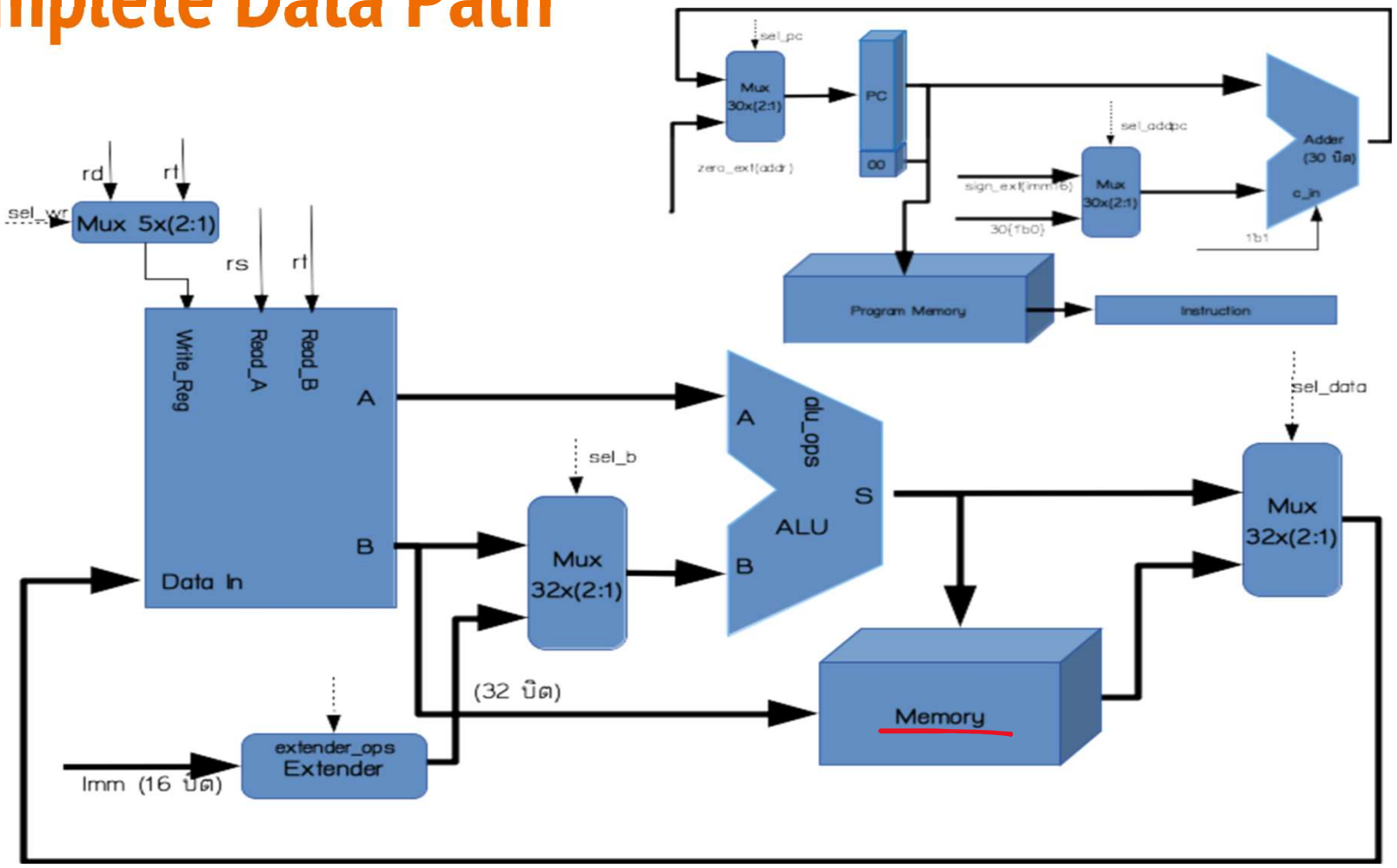
- Let's make an ALU that can
  - Add, Or, Sub

alu_ops	operations
00	S<- A+B
01	S<- A|B
10	S<- A-B

Step 3: Assembly Data Path

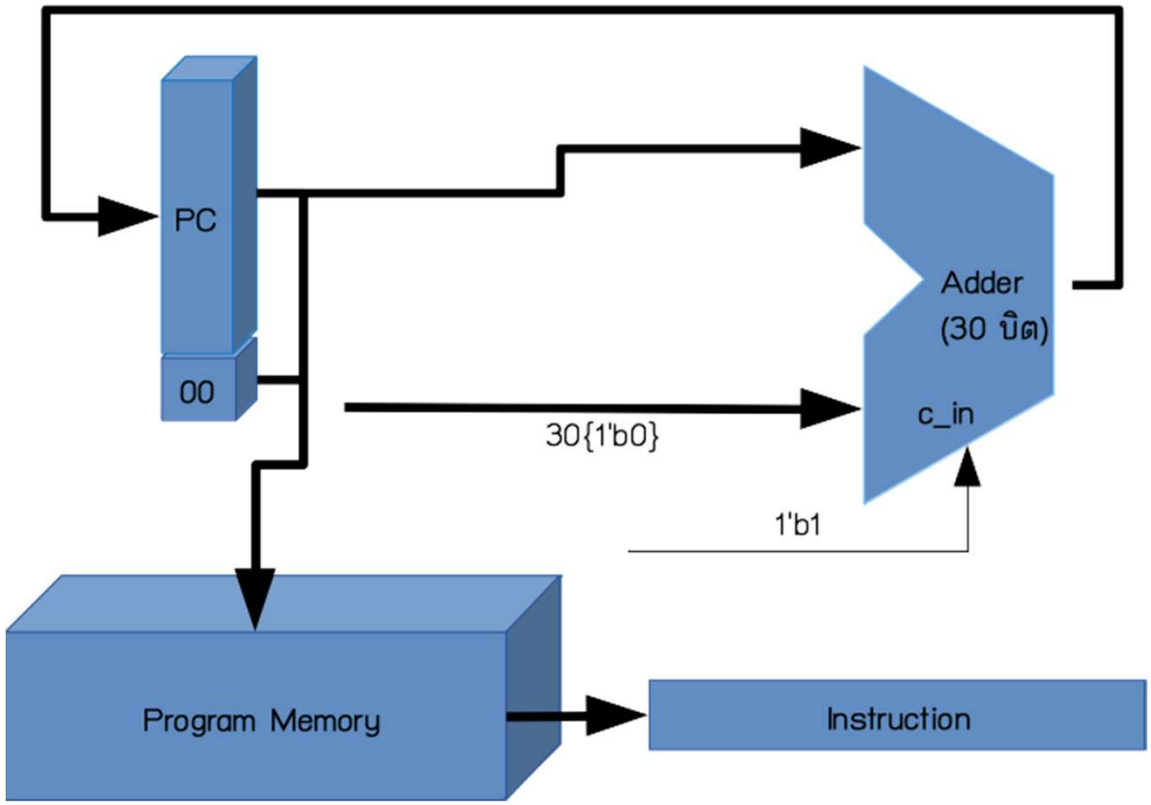
- ★ To assembly data path, simple add one instruction at a time
- ★ For each instruction added, modify the data path to support the RTL of the instruction.

Complete Data Path



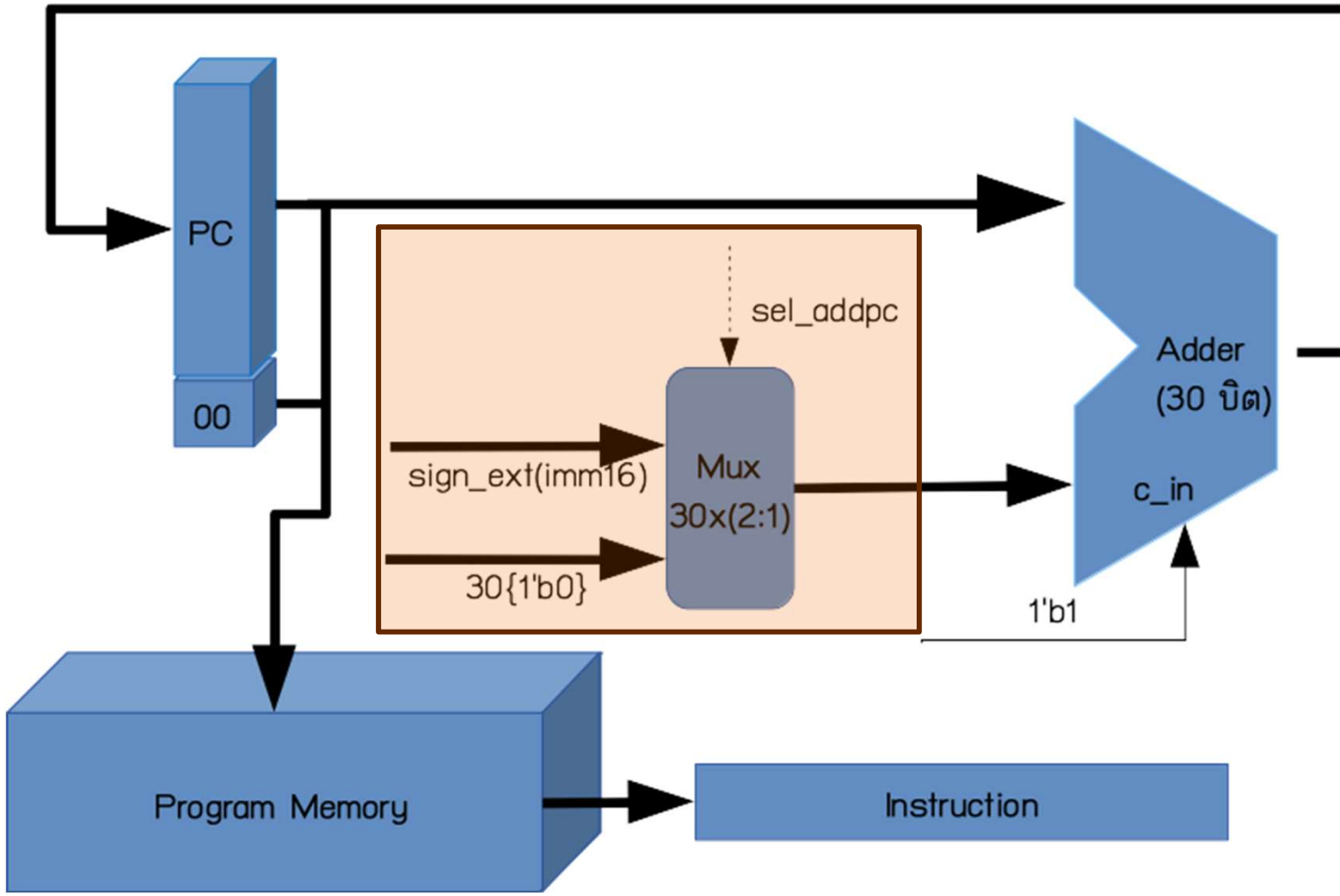
Fetch and PC

- ★ Fetch is usually ignored by RTL.
  - Instruction<- MEM[PC]
  - PC<- PC+4
- ★ Note.
  PC<-PC+4
  also means
  PC<-{PC[31:2]+1 , 2'b00}
  We can use 30-bit adder for it.



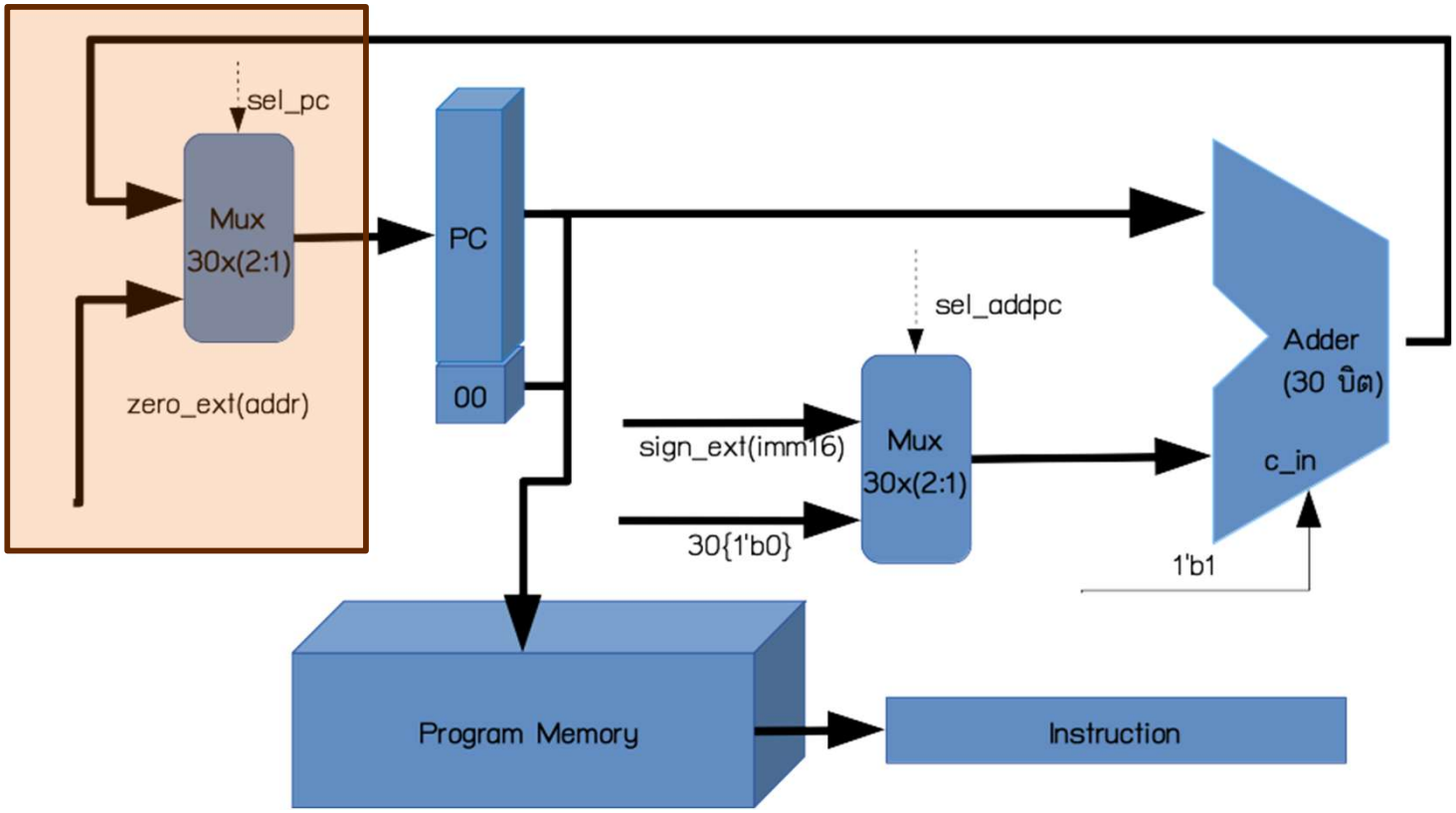
BEQ

- ★ BEQrs, rt, imm*4 ; (I-type)
  - If (R[rs] == R[rt]) then
  - PC ← PC + 4 + (sign_ext(imm) * 4)
  - else
  - PC ← PC + 4



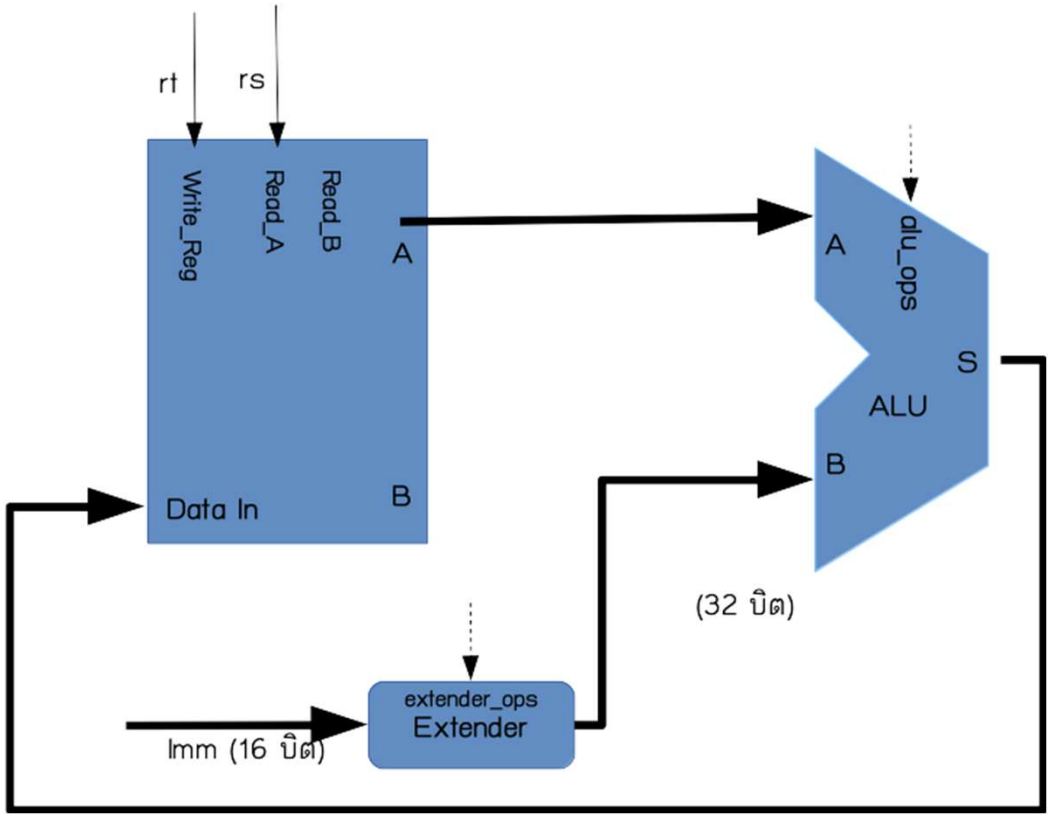
JMP

★ JMP addr*4 ; (J-type)
- PC ← (addr * 4)



ORI, ORUI

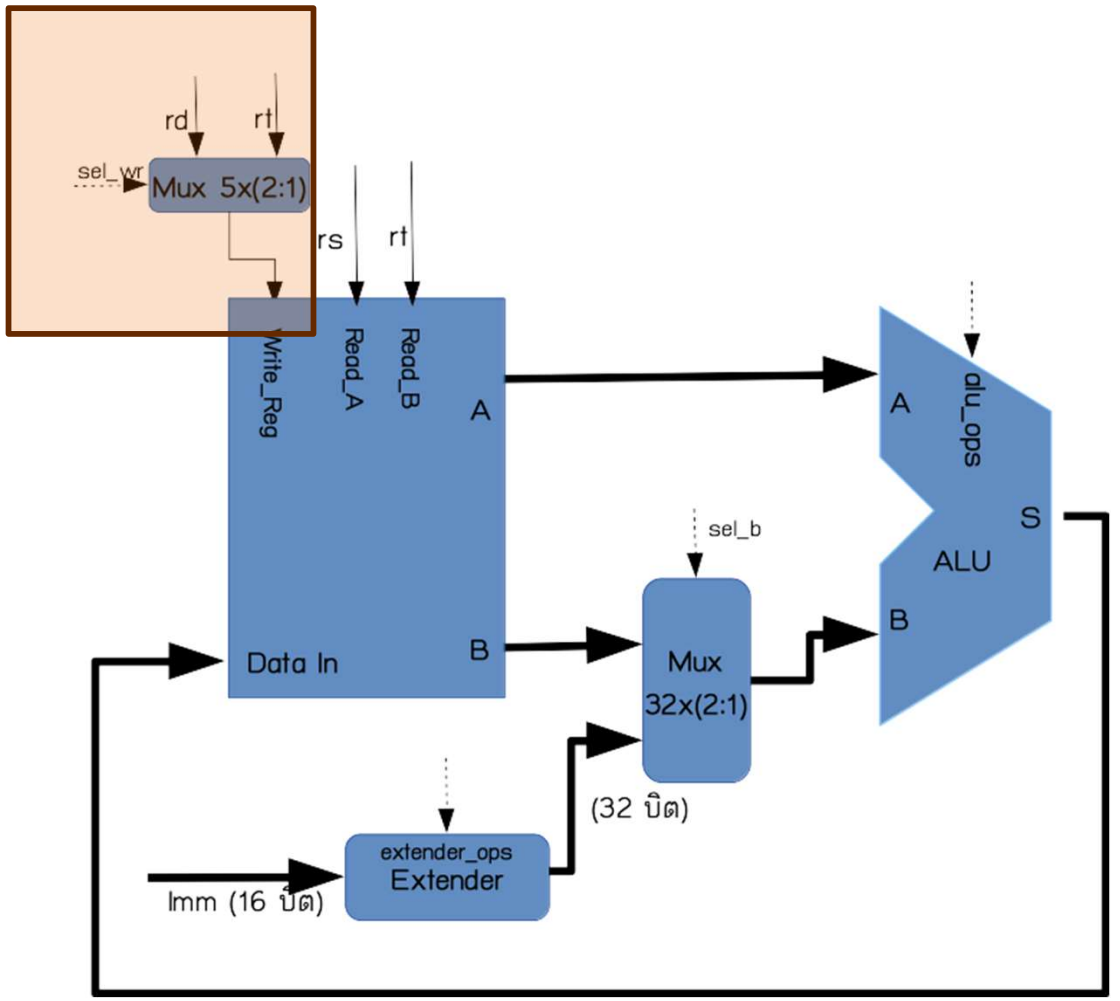
- ORI rt, rs, imm ; (I-type)
  - R[rt] ← R[rs] | zero_ext(imm);
    PC ← PC + 4
- ORUI rt, rs, imm
  ; (I-type)
  - R[rt] ← R[rs] | zero_pad(imm);
    PC ← PC + 4
- Same data path, only different extender operation.



Add

★ ADD rd, rs, rt ; (R-type)

- R[rd] ← R[rs] + R[rt];
  PC ← PC + 4



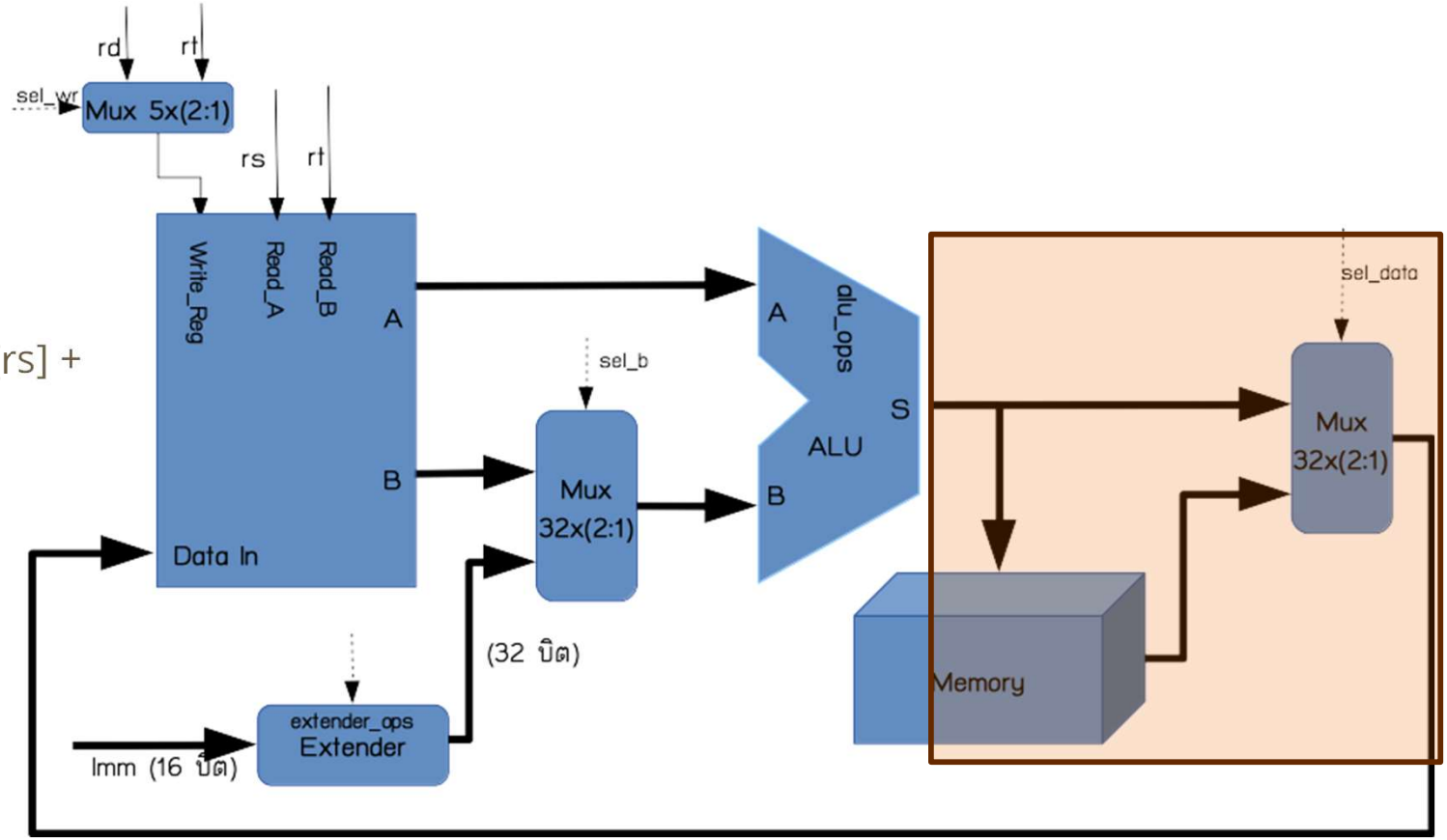
LW

★ LW rt, imm(rs)

; (I-type)

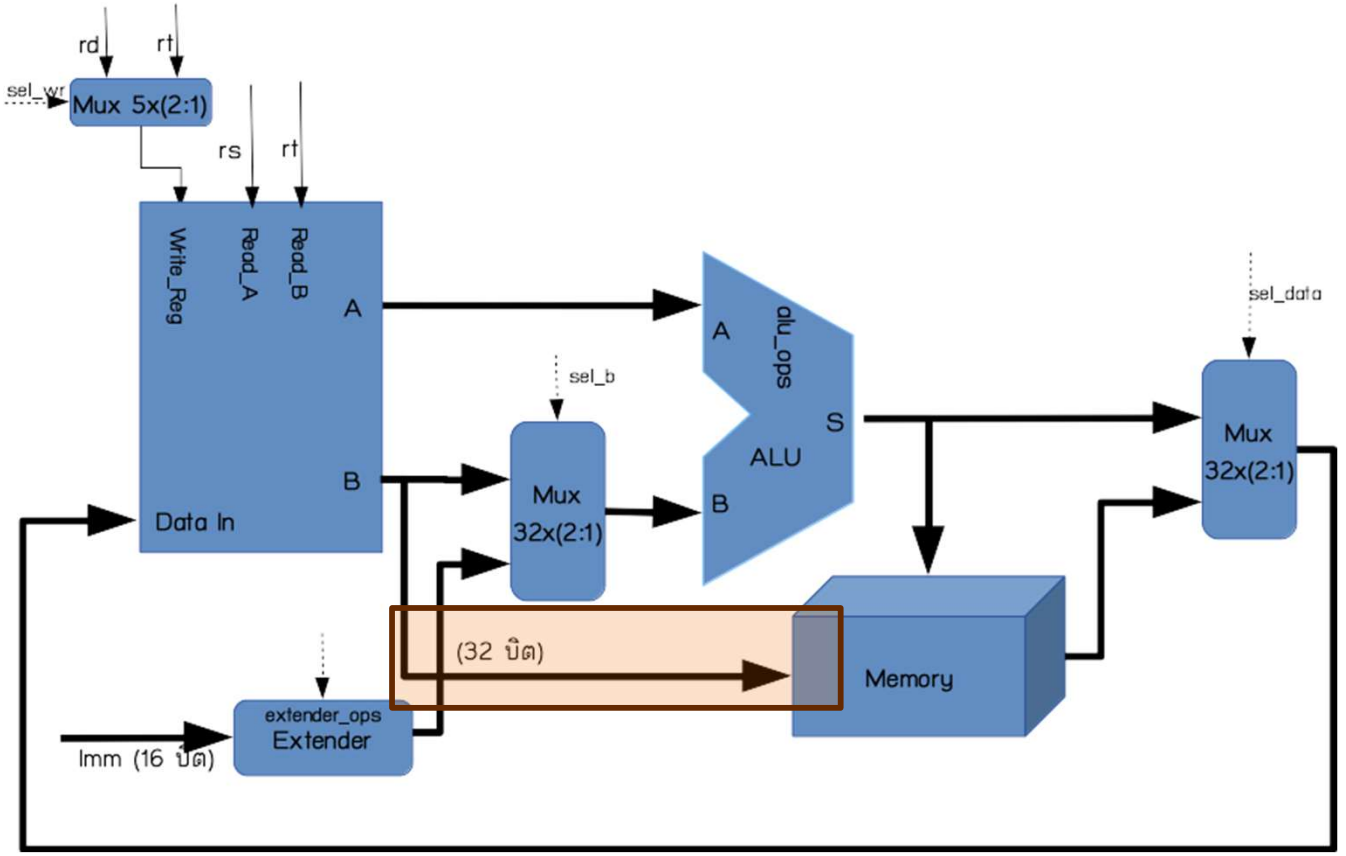
- R[rt] ← MEM[R[rs] + sign_ext(imm)];

  PC ← PC + 4

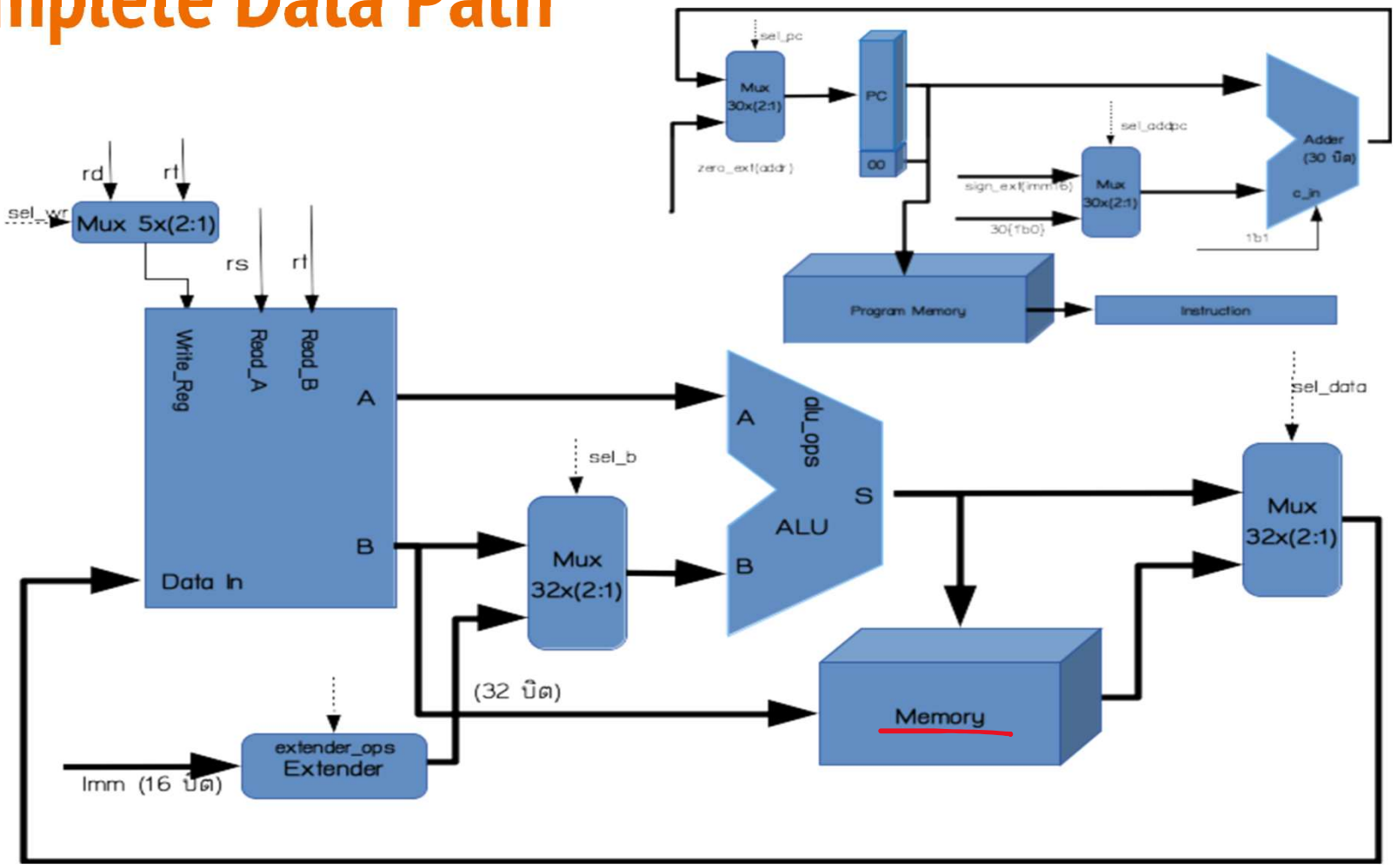


SW

- ★ SW rt, imm(rs) ; (I-type)
  - MEM[R[rs] + sign_ext(imm)] ← R[rt];
  - PC ← PC + 4



Complete Data Path



Step 4: RTL > Control (for the given datapath)

- Control Signals
  - sel_pc
  - sel_addpc
  - sel_wr
  - sel_b
  - sel_data
  - reg_wr
  - Mem_wr
  - alu_ops

Step 5: Control

Opcode	Zero (ALU)	sel_pc 0 – PC 1 - addr	sel_addpc 0 – 0 1 - addr	sel_wr 0 – rd 1 – rt	sel_b 0 – R[rt] 1 - imm	sel_data 0 – S 1 - M	reg_wr 1 – wr	mem_wr 1 – wr	alu_ops
ORI 010000	x	0	0	1	0	0	0	0	01
ORUI 010001	x	0	0	1	0	0	0	0	01
ADD 000001	x	0	0	0	0	0	0	0	00
LW 011000	x	0	0	1	1	1	0	0	00
SW 01100	x	0	0	x	1	x	0	1	00
BEQ 100100	0 1	0 0	0 1	x	x	x	0	0	10
JMP 110000	x	1	x	x	x	x	0	0	00

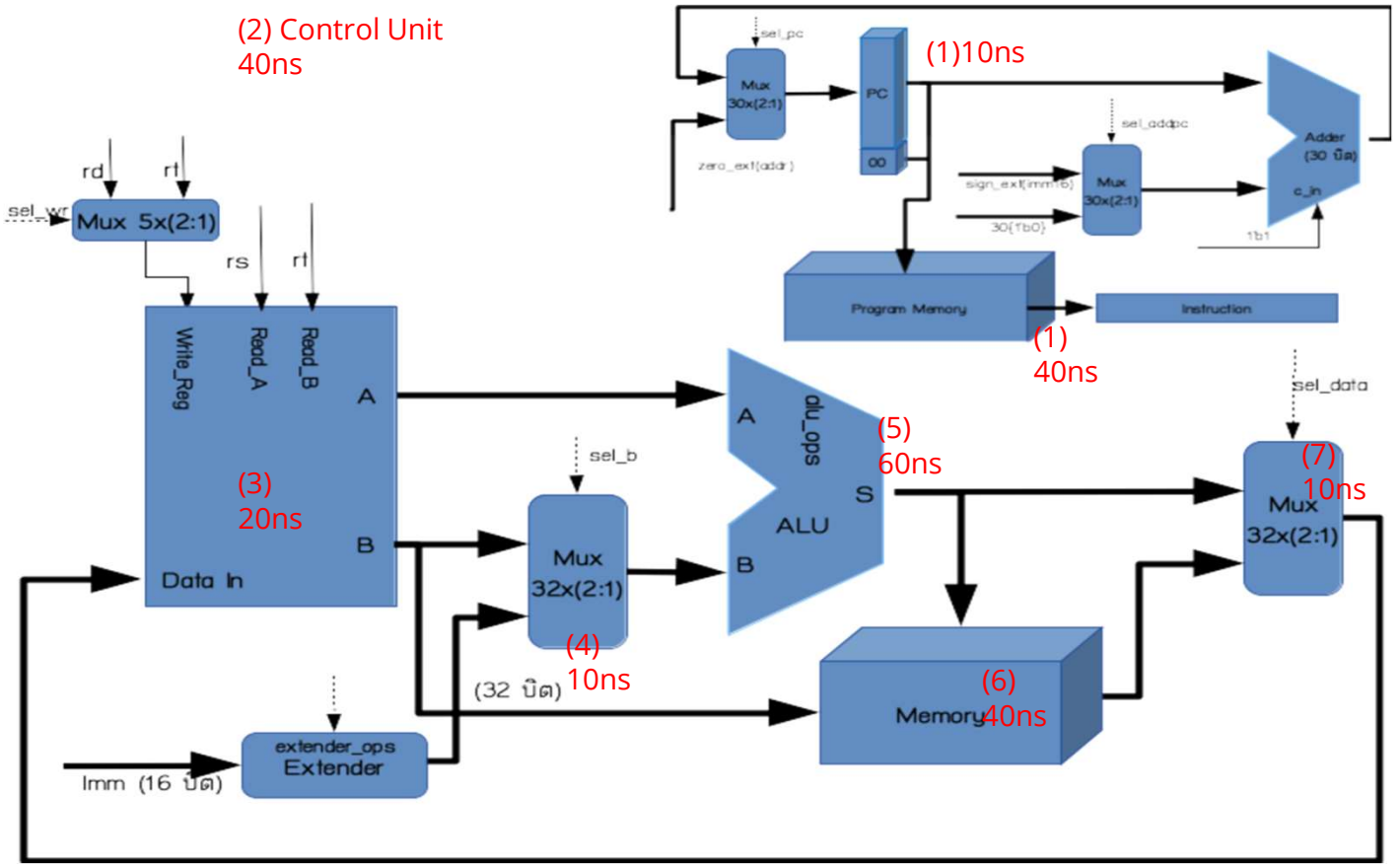
Control Implementation

- ★ Combinational Logic
- ★ ROM
- ★ PLA

Critical Path

(1) 10 + 40 + (2) 40 + (3) 20 + (4) 10 + (5) 60 + (6) 40 + (7) 10

- ★ Longest path?
- ★ Instruction ← MEM[PC]
  - PC 10ns
  - MEM 40ns
- ★ Control/Decode 40ns
- ★ Register 20ns
- ★ Extender 10ns
- ★ MUX 10ns
- ★ ALU 60ns
- ★ MUX 10ns



Summary

- ★ The simplicity of nanoLADA makes it easy.
  - Few instructions
  - Fixed-Length instruction set
  - Immediate and Location (Displacement) share the same field.
- ★ Single Cycle datapath
  - CPI=1
  - Cycle Time - Long
- ★ How to make it faster?

Given an ISA, can you design datapath?

Exercises

Performance

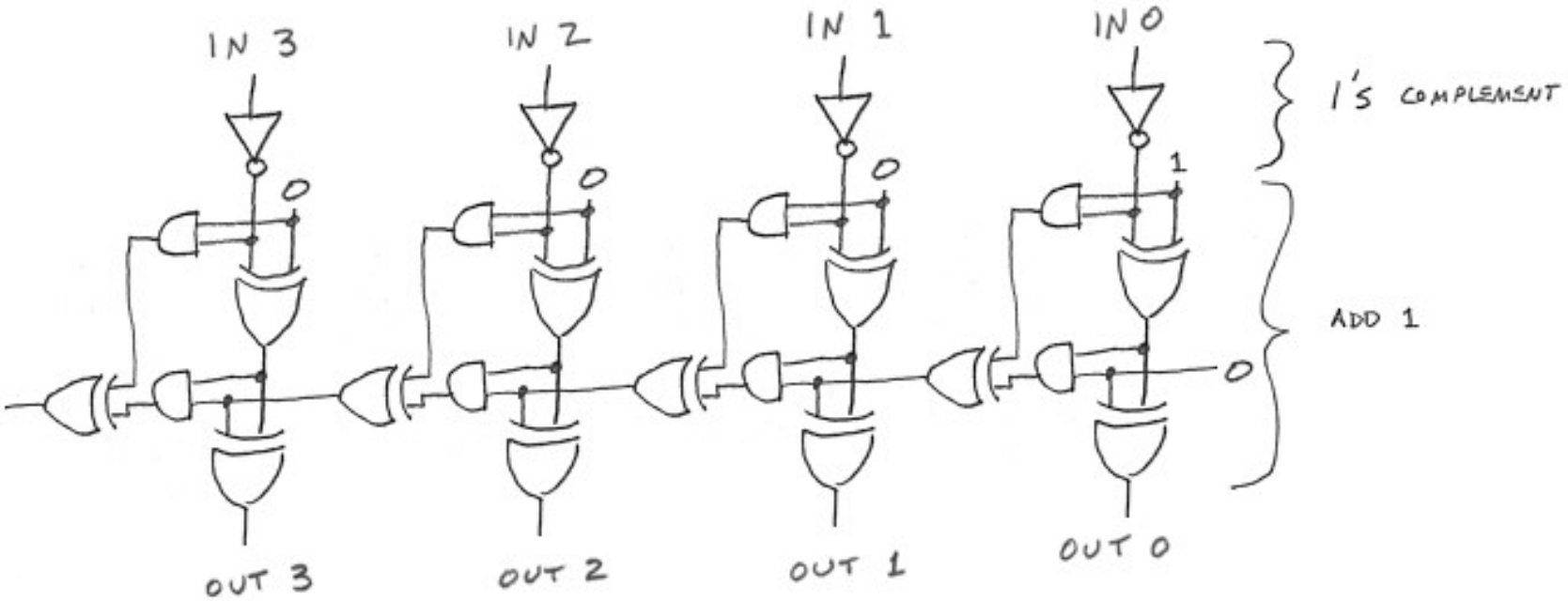
★ For the given single cycle processor, if a benchmark contains 2 millions instructions, calculate the CPU time.
Assuming that the critical path is 230 ns.

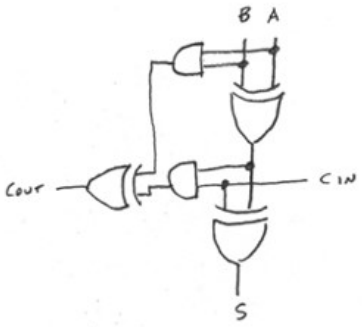
$$clock\ rate = \frac{10^9}{230} = 4.35 \times 10^6\ Hz$$

ALU

★ Please design an ALU that supports the following operations

alu_ops	Operation
00	S ← A + B ; z ← (S==0)
01	S ← A | B ; z ← (S==0)
10	S ← A – B ; z ← (S==0)





https://cs.stackexchange.com/questions/51034/twos-complement-using-only-logic-gates

PLA Implementation

★ How to implement the control signal (in p.26) with PLA?

G1 represents ORI (010000) - $G1=x5^{-}x4x3^{-}x2^{-}x1^{-}x0^{-}$

G2 represents ORUI (010001) - $G2=x5^{-}x4x3^{-}x2^{-}x1^{-}x0$

G3 represents ADD (000001) - $G3=x5^{-}x4^{-}x3^{-}x2^{-}x1^{-}x0$

G4 represents LW (011000) - $G4=x5^{-}x4x3x2^{-}x1^{-}x0^{-}$

G5 represents SW (011100) - $G5=x5^{-}x4x3x2x1^{-}x0^{-}$

G6 represents BEQ (100100) - $G6=x5x4^{-}x3^{-}x2x1^{-}x0^{-}$

G7 represents JMP (110000) - $G7=x5x4x3^{-}x2^{-}x1^{-}x0^{-}$

$sel\_pc=G7$

$sel\_addpc=G6+zero$

$sel\_wr=G1+G2+G4$

$sel\_b=G1+G2+G4+G5$

$sel\_data=G4$

$reg\_wr=G1+G2+G3+G4$

$mem\_wr=G5$

Adding scale operation

- Scale LD $r1, ($r2)[$r3*4]

End of Chapter 4