Computer System Architecture

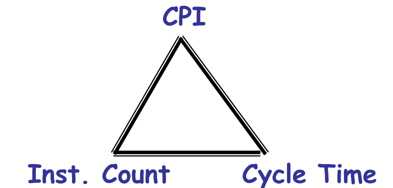
Processor Part I

Chalermek Intanagonwiwat

Slides courtesy of John Hennessy and David Patterson

The Big Picture: The Performance Perspective

- Performance of a machine is determined by:
  - Instruction count
  - Clock cycle time
  - Clock cycles per instruction



The Big Picture (cont.)

- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction
- Today:
  - Single cycle processor:
    - Advantage: One clock cycle per instruction
    - Disadvantage: long cycle time

How to Design a Processor: step-by-step

1. Analyze instruction set => datapath <u>requirements</u>
   - the meaning of each instruction is given by the *Register Transfer Language (RTL)*
   - datapath must include storage element for ISA registers
     - possibly more
   - datapath must support each register transfer

How to Design a Processor (cont.)

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

The MIPS Instruction Formats

- All MIPS (Microprocessor without Interlocked Pipeline Stages) instructions are 32 bits long.
- The three instruction formats:

– R-type

31–26	25–21	20–16	15–11	10–6	5–0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

– I-type

31–26	25–21	20–16	15–0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

– J-type

31–26	25–0
op	target address
6 bits	26 bits

The MIPS Instruction Formats (cont.)

- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: target address of the jump instruction

Step 1a: The MIPS-lite Subset

- ADD and SUB
  - addU rd, rs, rt
  - subU rd, rs, rt

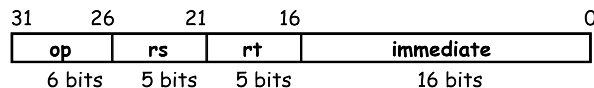
31–26	25–21	20–16	15–11	10–6	5–0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- OR Immediate:
  - ori rt, rs, imm16

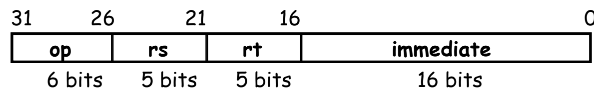
31–26	25–21	20–16	15–0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

Step 1a: The MIPS-lite Subset (cont.)

- LOAD and STORE Word
  - lw rt, rs, imm16
  - sw rt, rs, imm16



- BRANCH:
  - beq rs, rt, imm16



Logical Register Transfers

- RTL gives the <u>meaning</u> of the instructions
- All start by fetching the instruction

Logical Register Transfers (cont.)

op | rs | rt | rd | shamt | funct = MEM[PC]

op | rs | rt | Imm16 = MEM[PC]

inst	Register Transfers	
ADDU	R[rd] <- R[rs] + R[rt];	PC <- PC + 4
SUBU	R[rd] <- R[rs] – R[rt];	PC <- PC + 4
ORi	R[rt] <- R[rs] | zero_ext(Imm16);	PC <- PC + 4
LOAD	R[rt] <- MEM[R[rs] + sign_ext(Imm16)];	PC <- PC + 4
STORE	MEM[R[rs] + sign_ext(Imm16)] <- R[rt];	PC <- PC + 4
BEQ	if (R[rs] == R[rt]) then PC <- PC + sign_ext(Imm16)] || 00 else PC <- PC + 4	

Step 1: Requirements of the Instruction Set

- Memory
  - instruction & data
- Registers (32 x 32)
  - read RS
  - read RT
  - Write RT or RD
- PC

Step 1: Requirements of the Instruction Set (cont.)

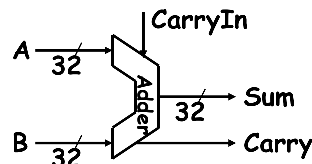
- Extender
- Add and Sub register or extended immediate
- Add 4 or extended immediate to PC

Step 2: Components of the Datapath

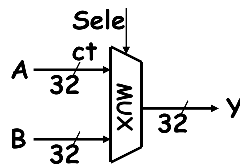
- Combinational Elements
- Storage Elements
  - Clocking methodology

Combinational Logic Elements

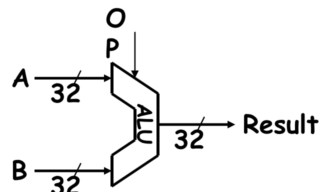
- Adder



- MUX

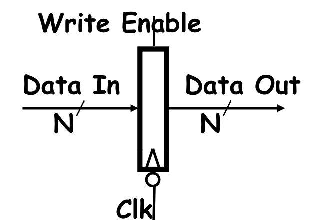


- ALU



Storage Element: Register

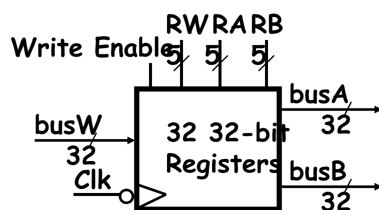
- Similar to the D Flip Flop except
  - N-bit input and output
  - Write Enable input



- Write Enable:
  - negated (0): Data Out will not change
  - asserted (1): Data Out will become Data In

Register File

- Register File consists of 32 registers:
  - Two 32-bit output busses: busA and busB
  - One 32-bit input bus: busW



Register File (cont.)

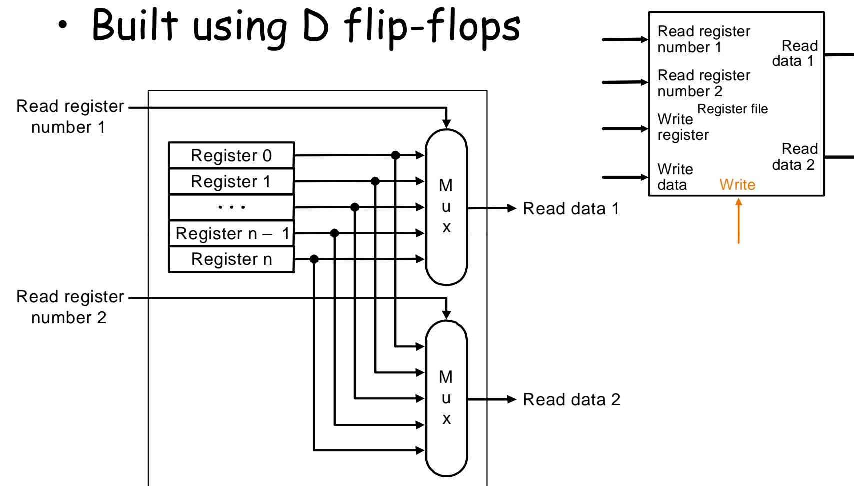
- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1

Register File (cont.)

- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid => busA or busB valid after "access time."

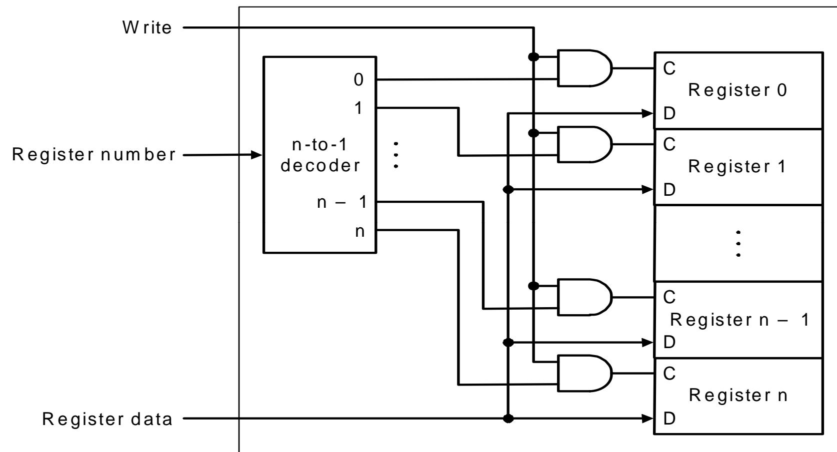
Register File (cont.)

- Built using D flip-flops



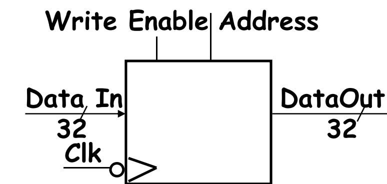
Register File (cont.)

- Note: we still use the real clock to determine when to write



Storage Element: Idealized Memory

- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out



Storage Element: Idealized Memory (cont.)

- Memory word is selected by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory
    word to be written via the Data In bus

Storage Element: Idealized Memory (cont.)

- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - Address valid => Data Out valid after "access time."

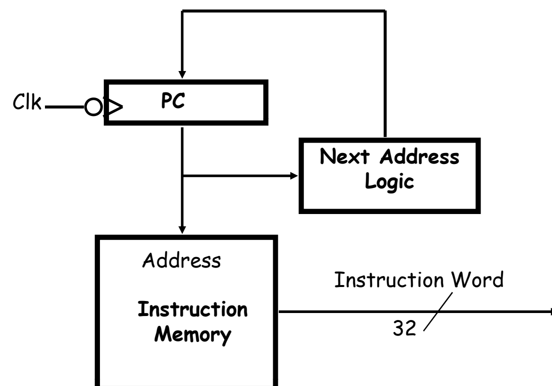
Step 3

- Register Transfer Requirements –> Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation

3a: Overview of the Instruction Fetch Unit

- The common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump: PC <- "something else"

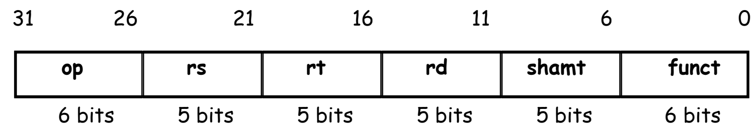
3a: Overview of the Instruction Fetch Unit (cont.)

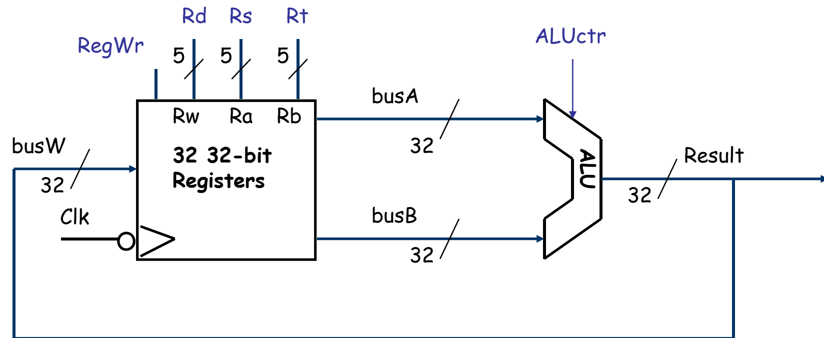


3b: Add & Subtract

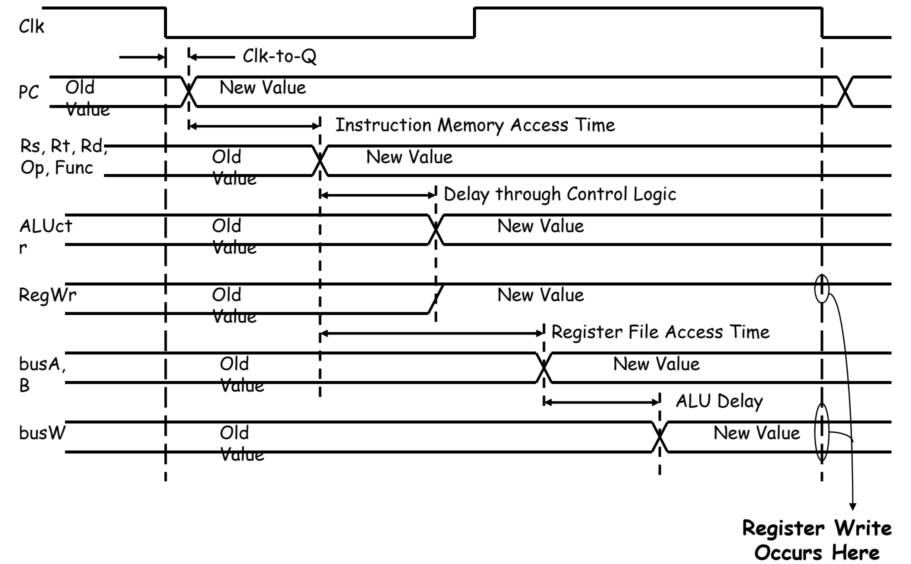
- R[rd] <- R[rs] op R[rt]
  Example: addU rd, rs, rt
  - Ra, Rb, and Rw come from instruction's rs, rt, and rd fields
  - ALUctr and RegWr: control logic after decoding the instruction

3b: Add & Subtract (cont.)



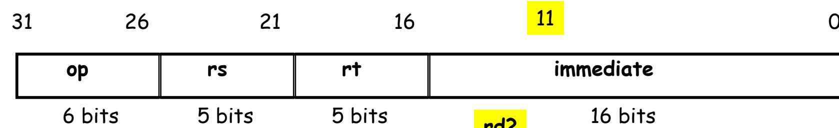


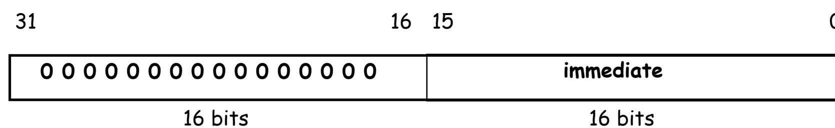
Register-Register Timing



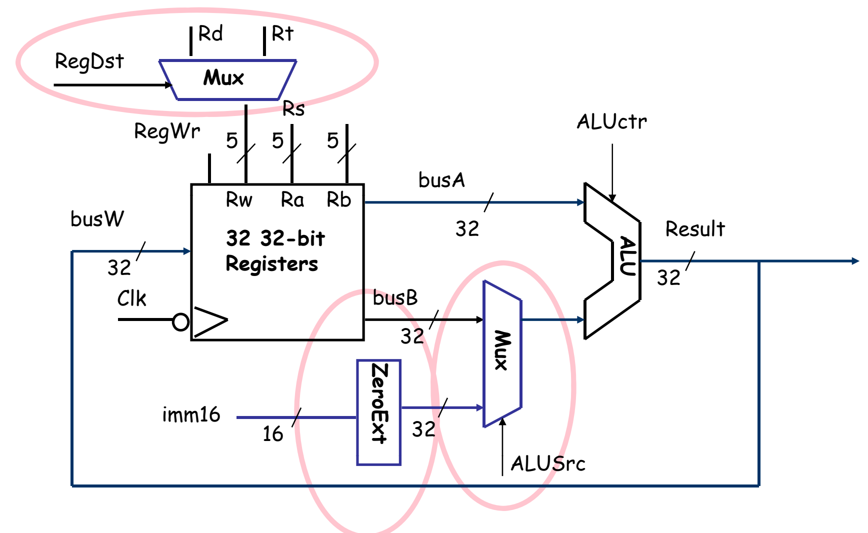
3c: Logical Operations with Immediate

- R[rt] <- R[rs] op ZeroExt[imm16]]



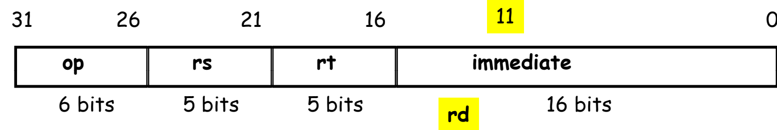


3c: Logical Operations with Immediate (cont.)

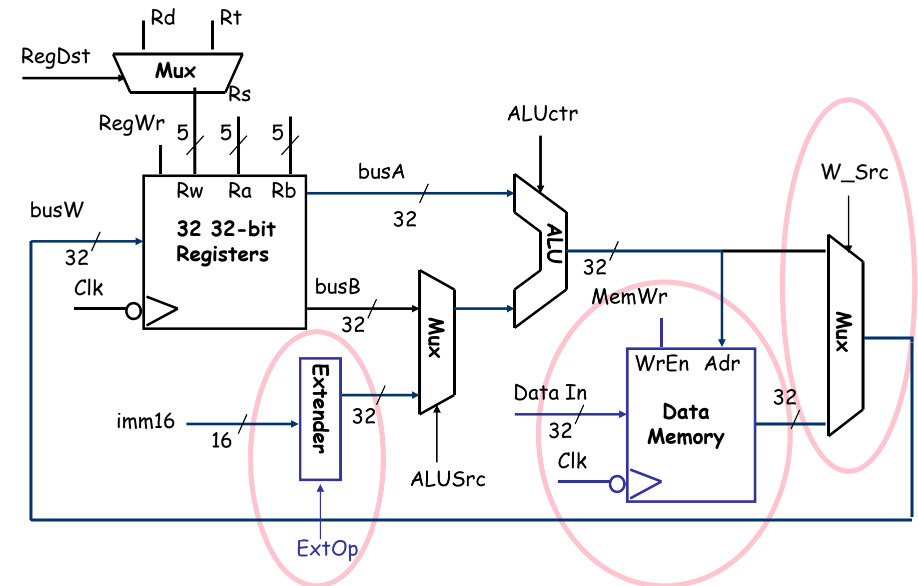


3d: Load Operations

- R[rt] <- Mem[R[rs] + SignExt[imm16]]
  Example: lw rt, rs, imm16

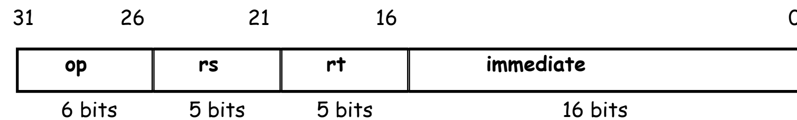


3d: Load Operations (cont.)

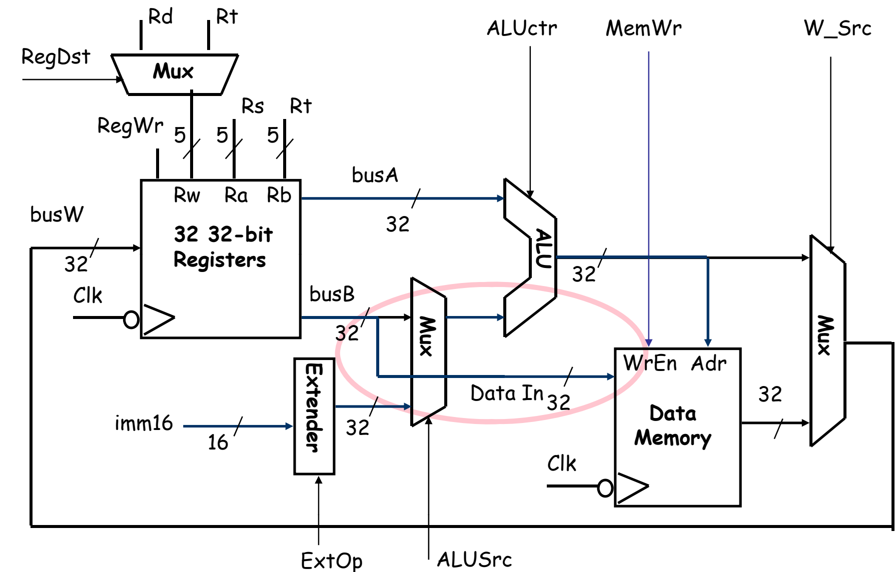


3e: Store Operations

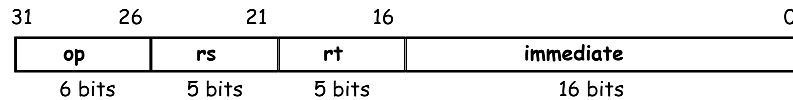
- Mem[R[rs] + SignExt[imm16]] <- R[rt]
  Example: sw rt, rs, imm16



3e: Store Operations (cont.)

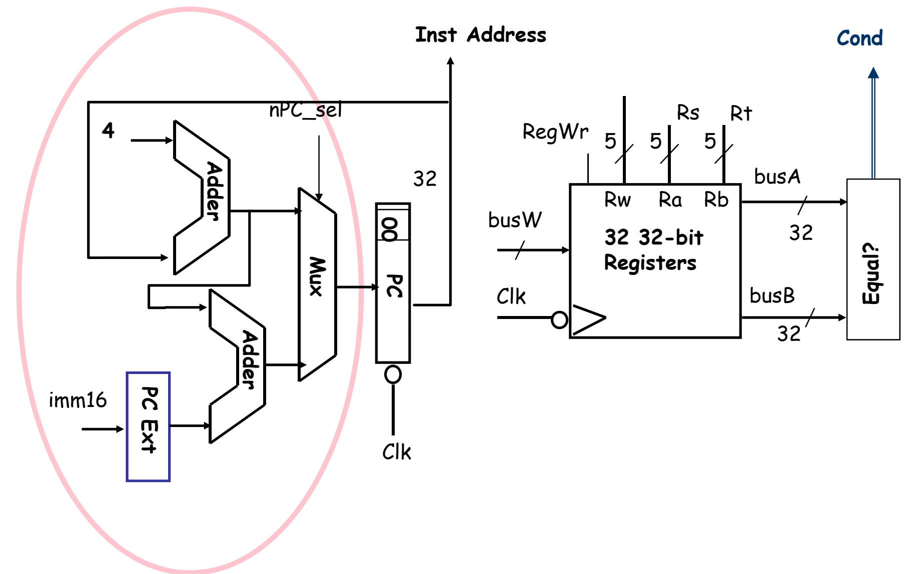


3f: The Branch Instruction

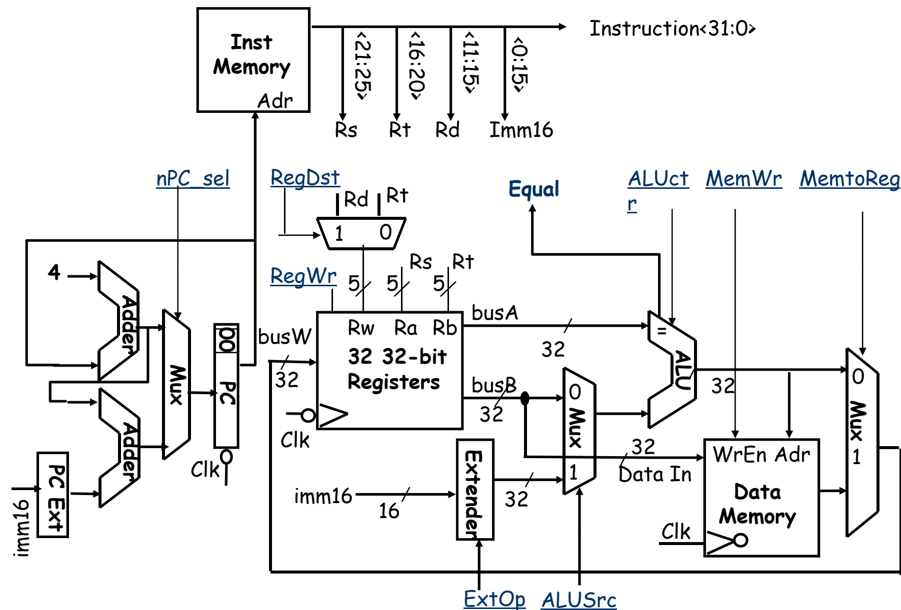


- beq rs, rt, imm16
  - mem[PC] from memory — Fetch the instruction
  - Equal <- R[rs] == R[rt] — Calculate the branch condition
  - if (COND eq 0) — Calculate the next instruction's address
    - PC <- PC + 4 + (SignExt(imm16) x 4)
  - else
    - PC <- PC + 4

Datapath for Branch Operations



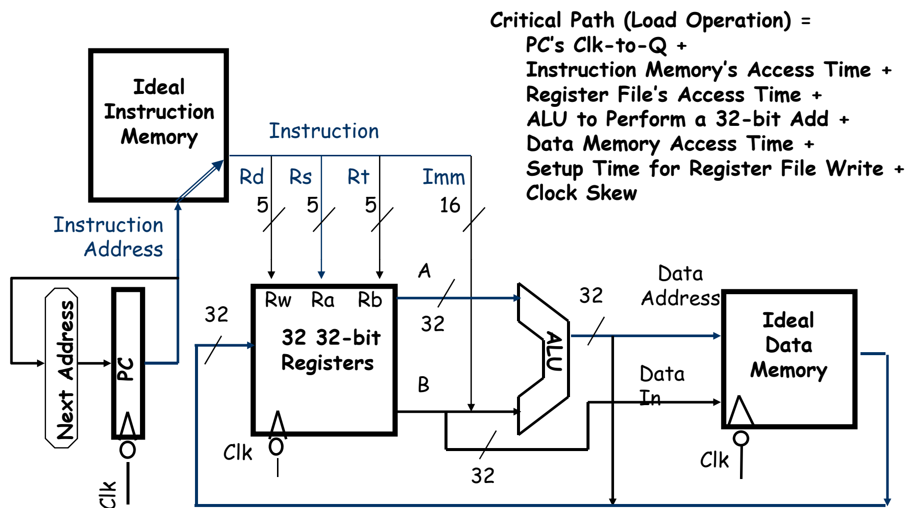
A Single Cycle Datapath



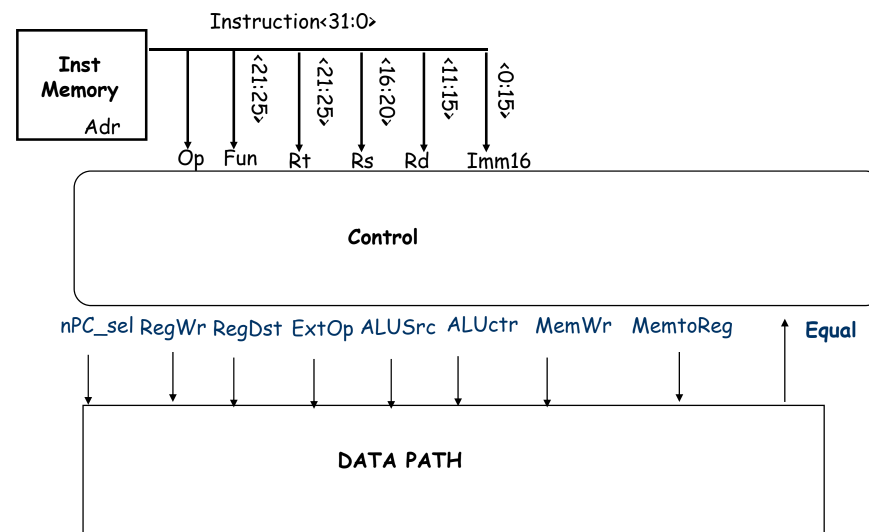
An Abstract View of the Critical Path

- Register file and ideal memory:
  - The CLK input is a factor ONLY during write operation
  - During read operation, behave as combinational logic:
    - Address valid => Output valid after "access time."

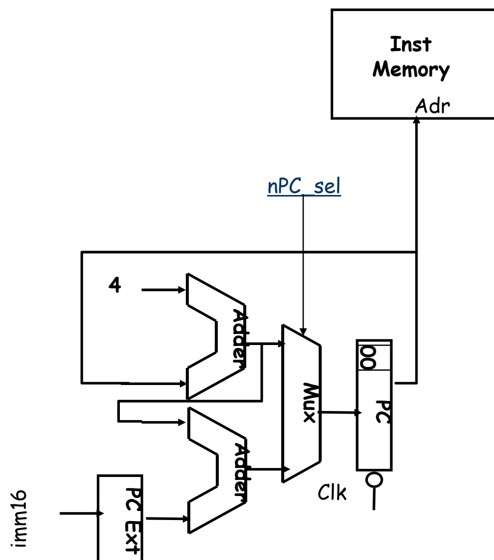
An Abstract View of the Critical Path (cont.)



Step 4: Given Datapath: RTL -> Control



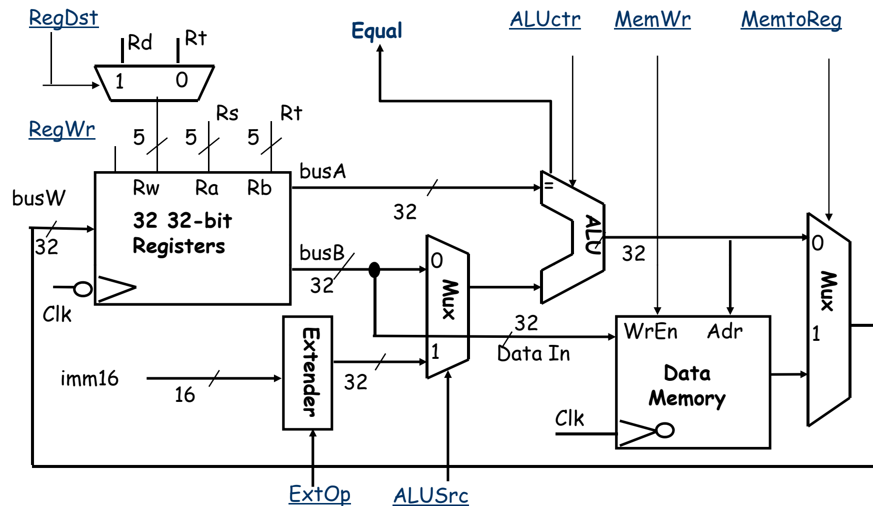
Meaning of the Control Signals



Meaning of the Control Signals (cont.)

- Rs, Rt, Rd and Imed16 hardwired into datapath
- nPC_sel:
  - 0 => PC <– PC + 4;
  - 1 => PC <– PC + 4 + SignExt(Im16) || 00

Meaning of the Control Signals (cont.)



Meaning of the Control Signals (cont.)

- ExtOp: "zero", "sign"
- ALUsrc: 0 => regB; 1 => immed
- ALUctr: "add", "sub", "or"
- MemWr: write memory
- MemtoReg: 1 => Mem
- RegDst: 0 => "rt"; 1 => "rd"
- RegWr: write dest register

Control Signals (cont.)

<u>inst</u> <u>Register Transfer</u>

ADD R[rd] <– R[rs] + R[rt]; PC <– PC + 4

ALUsrc = RegB, ALUctr = "add", RegDst = rd, RegWr, nPC_sel = "+4"

SUB R[rd] <– R[rs] – R[rt]; PC <– PC + 4

ALUsrc = RegB, ALUctr = "sub", RegDst = rd, RegWr, nPC_sel = "+4"

ORi R[rt] <– R[rs] + zero_ext(Imm16); PC <– PC + 4

ALUsrc = Im, Extop = "Z", ALUctr = "or", RegDst = rt, RegWr, nPC_sel = "+4"

Control Signals (cont.)

LOAD R[rt] <– MEM[R[rs] + sign_ext(Imm16)];

PC <– PC + 4

ALUsrc = Im, Extop = "Sn", ALUctr = "add", MemtoReg, RegDst = rt, RegWr, nPC_sel = "+4"

STORE MEM[R[rs] + sign_ext(Imm16)] <– R[rs];

PC <– PC + 4

ALUsrc = Im, Extop = "Sn", ALUctr = "add", MemWr, nPC_sel = "+4"

BEQ if (R[rs] == R[rt]) then

PC <– PC + sign_ext(Imm16)] || 00 else PC <– PC + 4

nPC_sel = EQUAL, ALUctr = "sub"

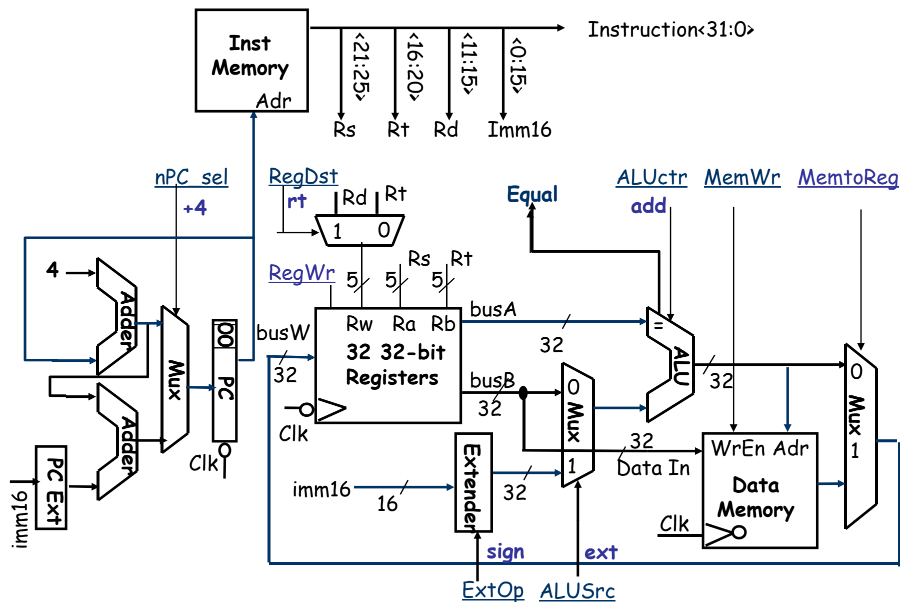
Step 5: Logic for each control signal

- nPC_sel <= if (OP == BEQ) then EQUAL else 0
- ALUsrc <= if (OP == R-TYPE) then "regB" else "immed"
- ALUctr <= if (OP == R-TYPE) then **funct**
  elseif (OP == ORi) then "OR"
  elseif (OP == BEQ) then "sub"
  else "add"

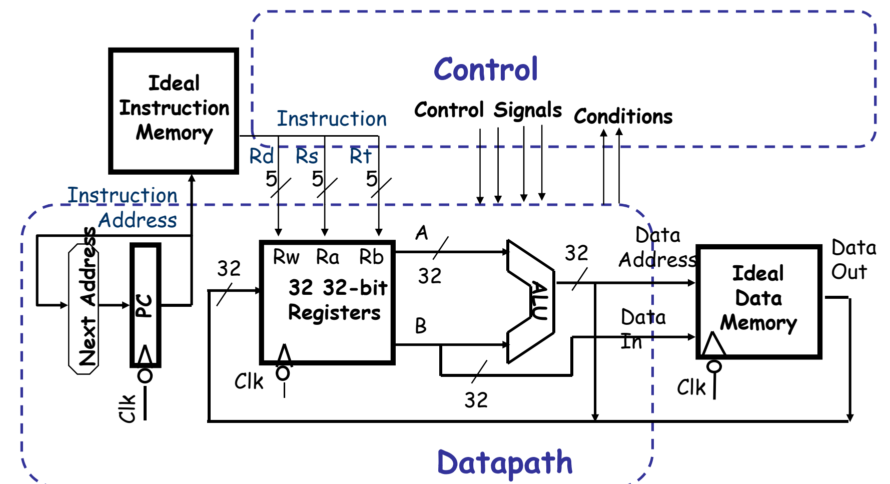
Step 5: Logic for each control signal (cont.)

- ExtOp <= if (OP == ORi) then "zero" else "sign"
- MemWr <= (OP == Store)
- MemtoReg <= (OP == Load)
- RegWr: <= if ((OP == Store) || (OP == BEQ)) then 0 else 1
- RegDst: <= if ((OP == Load) || (OP == ORi)) then 0 else 1

Example: Load Instruction



An Abstract View of the Implementation



- Logical vs. Physical Structure

Summary

- 5 steps to design a processor
  - 1. Analyze instruction set => datapath requirements
  - 2. Select set of datapath components & establish clock 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

Summary (cont.)

- MIPS makes it easier
  - Instructions same size
  - Source registers always in same place
  - Immediates same size, location
  - Operations always on registers/immediates
- Single cycle datapath => CPI=1, CCT => long
- Next time: implementing control