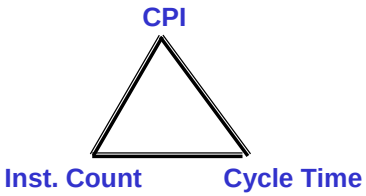
L07: Single Cycle Datapath Design

Adapted from:

CS152: Computer Architecture and Engineering
Dave Patterson (www.cs.berkeley.edu/~pattrsn)

The Big Picture: The Performance Perspective

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



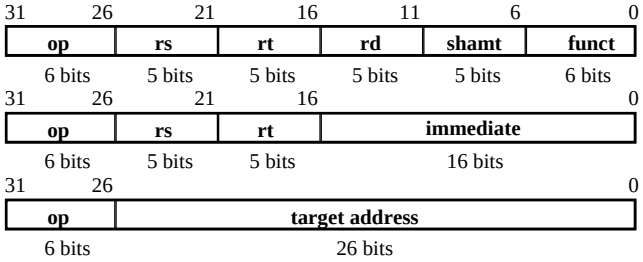
- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction
- Today:
  - Single cycle processor:
    - Advantage: One clock cycle per instruction
    - Disadvantage: long cycle time

How to Design a Processor: step-by-step

1. Analyze instruction set ⇒ datapath requirements
   - the meaning of each instruction is given by the register transfers
   - datapath must include storage element for ISA registers
   - datapath must support each register transfer
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 affect the register transfer.
5. Assemble the control logic

The MIPS Instruction Formats

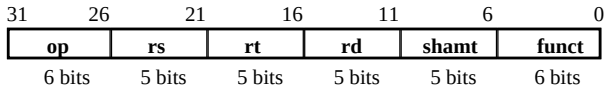
- All MIPS instructions are 32 bits long. The three instruction formats:



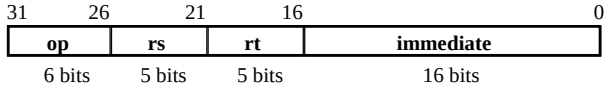
  - R-type
  - I-type
  - J-type
- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the “op” field
  - address / immediate: address offset or immediate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

Step 1a: The MIPS-lite Subset for today

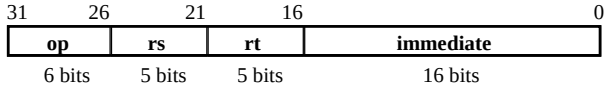
- **ADD and SUB**
  - **addU rd, rs, rt**
  - **subU rd, rs, rt**



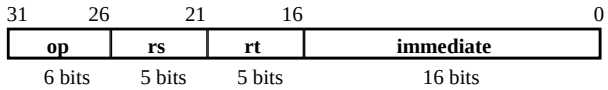
- **OR Immediate:**
  - **ori rt, rs, imm16**



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



- **BRANCH:**
  - **beq rs, rt, imm16**



Logical Register Transfers

- **RTL gives the meaning of the instructions**
- **All start by fetching the instruction**

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

op | rs | rt | Imm16

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 + 4 + sign_ext(Imm16) || 00 else PC ← PC + 4;

Step 1: Requirements of the Instruction Set

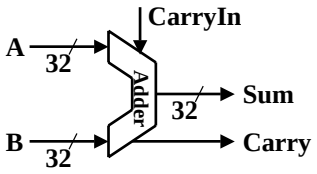
- **Memory**
  - **instruction & data**
- **Registers (let's say 32 x 32)**
  - **read RS**
  - **read RT**
  - **Write RT or RD**
- **PC**
- **Extender**
- **Add and Sub register or extended immediate**
- **Add 4 or extended immediate to PC**

Step 2: Components of the Datapath

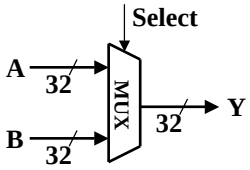
- **Combinational Elements**
- **Storage Elements**
  - **Clocking methodology**

Combinational Logic Elements (Basic Building Blocks)

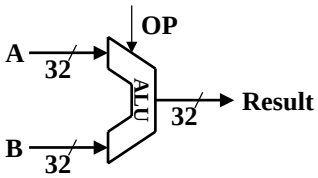
- **Adder**



- **MUX**

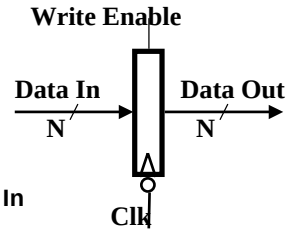


- **ALU**



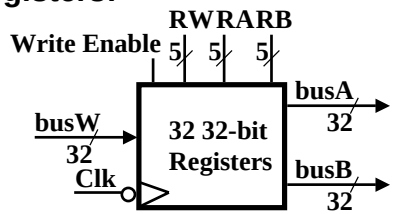
Storage Element: Register (Basic Building Block)

- **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**



Storage Element: Register File

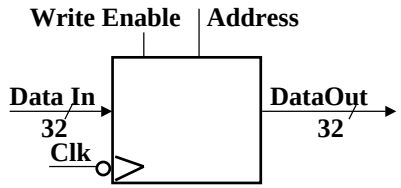
- **Register File consists of 32 registers:**
  - **Two 32-bit output busses:**
  - **busA and busB**
  - **One 32-bit input bus: busW**



- **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**
- **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."**

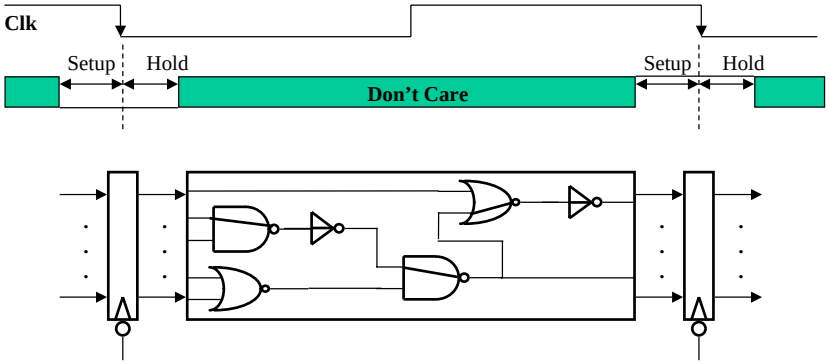
Storage Element: Memory

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



- **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**
- **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."**

Clocking Methodology



All storage elements are clocked by the same clock edge

Cycle Time ≥ CLK-to-Q + Longest Delay Path + Setup + Clock Skew

- **Clock skew = difference in arrival time of clock edges**

(CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time

Step 3

Register Transfer Requirements
→ Datapath Assembly

1. **Instruction Fetch**
2. **Read Operands and Execute Operation**

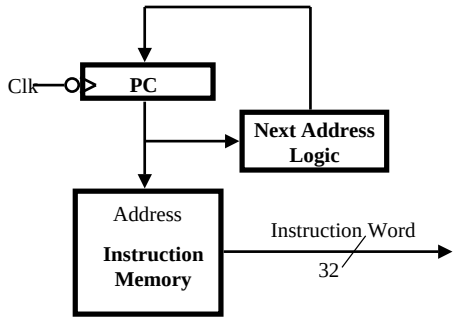
3a: Overview of the Instruction Fetch Unit

Common RTL operations:

a) **At start of cycle, fetch the instruction: mem[PC]**

b) **At end of cycle, update the program counter:**

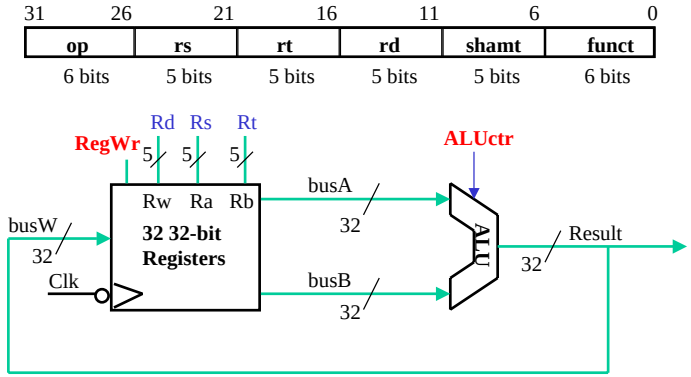
- **Sequential Code: PC ← PC + 4**
- **Branch and Jump: PC ← "something else"**



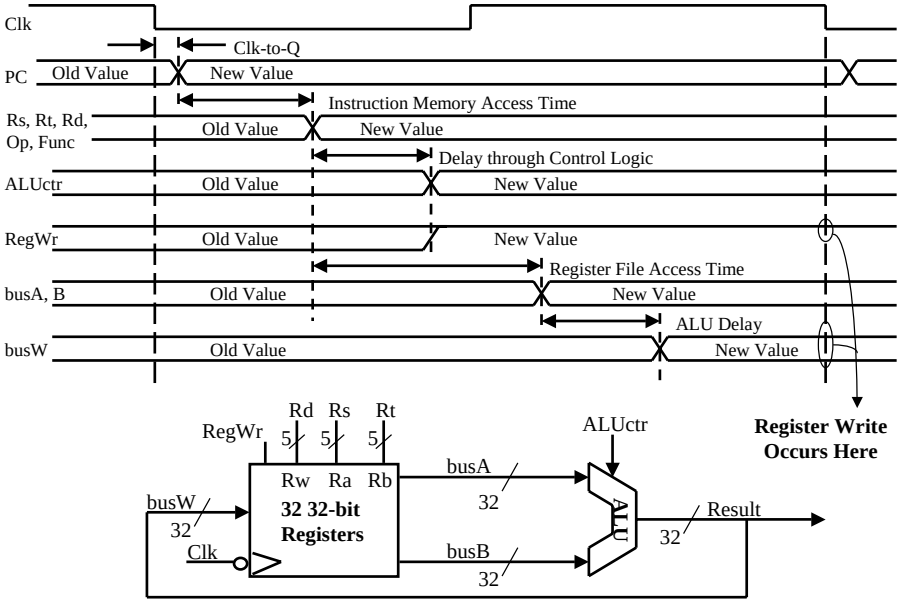
3b: Add & Subtract

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**

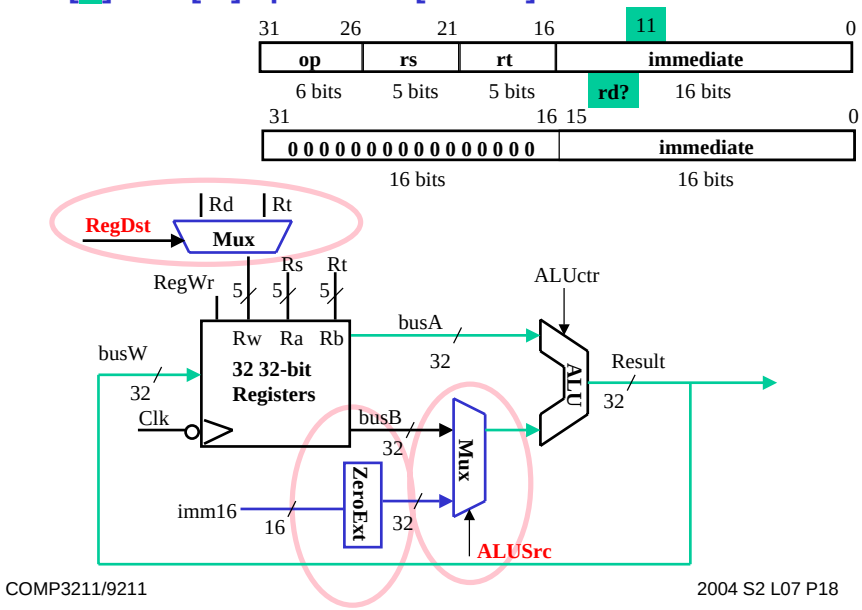


Register-Register Timing



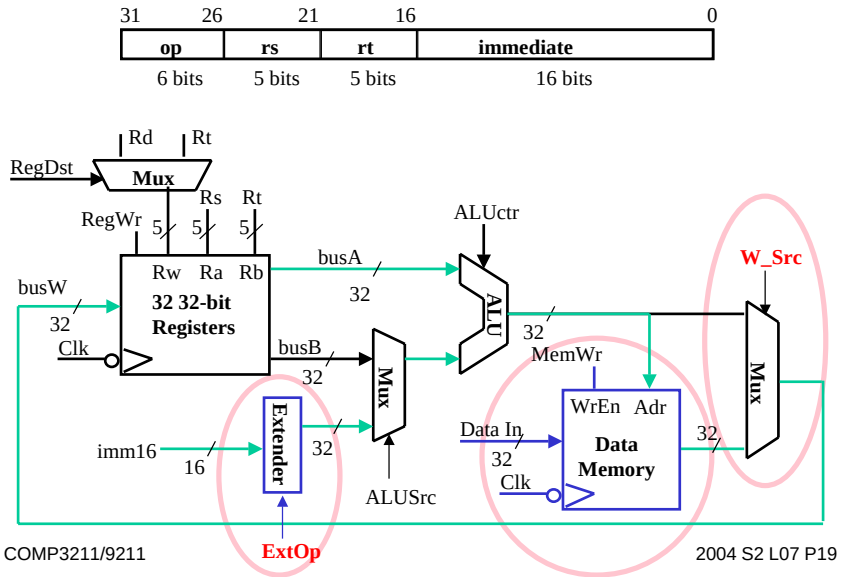
3c: Logical Operations with Immediate

R[rt] ← R[rs] op ZeroExt[imm16]



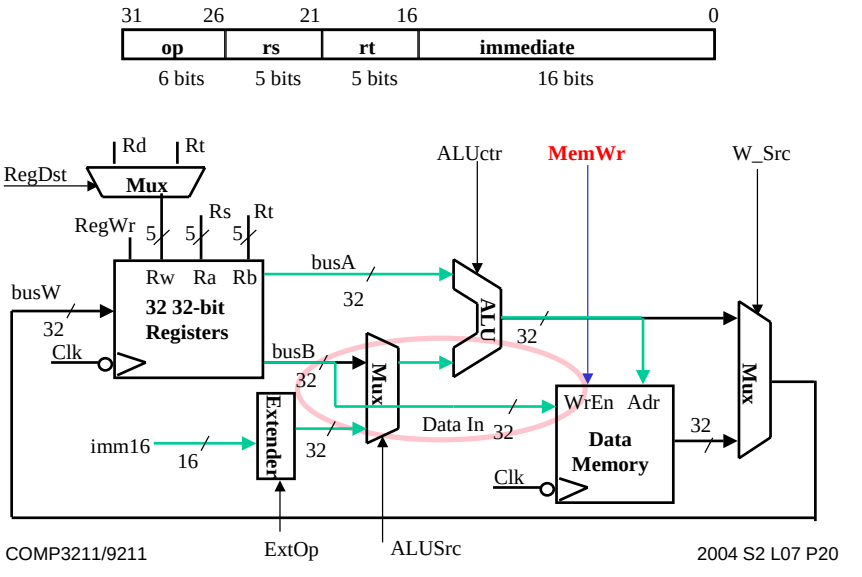
3d: Load Operations

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

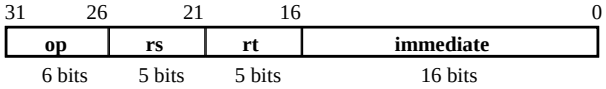


3e: Store Operations

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



3f: The Branch Instruction



beq rs, rt, imm16

- **mem[PC]** — Fetch the instruction from memory
- **Equal ← R[rs] == R[rt]** — Calculate the branch condition
- **if (COND eq 0)** — Calculate the next instruction's address

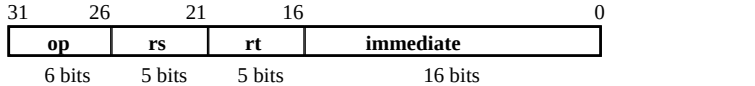
  PC ← PC + 4 + (SignExt(imm16) x 4)

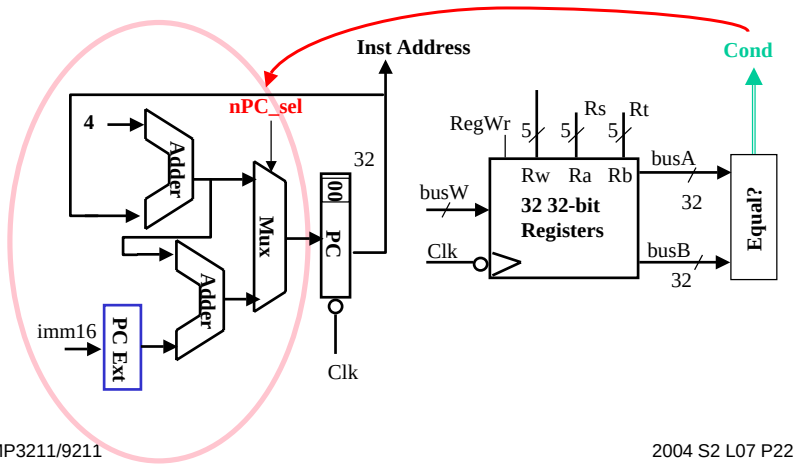
  else

  PC ← PC + 4

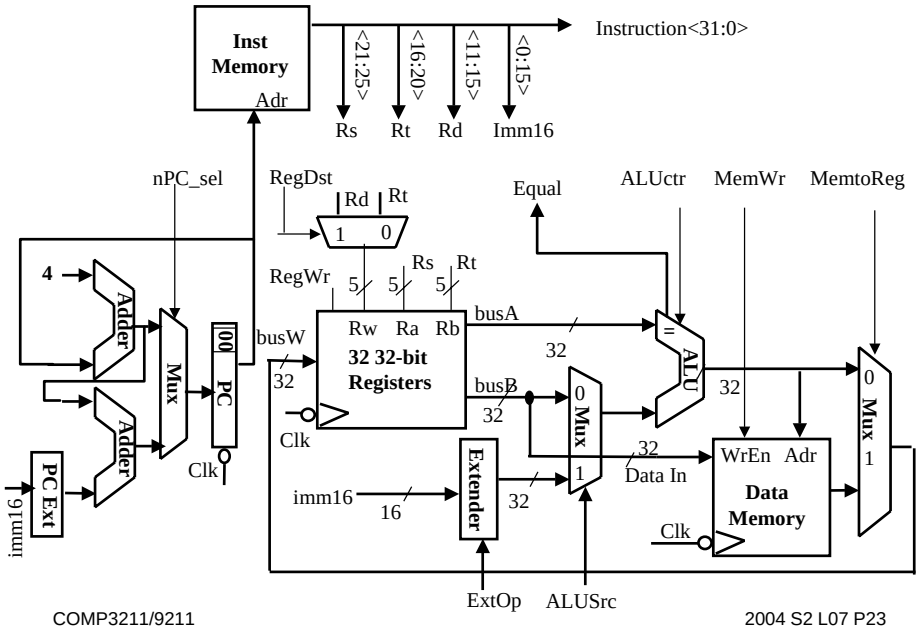
Datapath for Branch Operations

beq rs, rt, imm16 **Datapath generates condition (equal)**





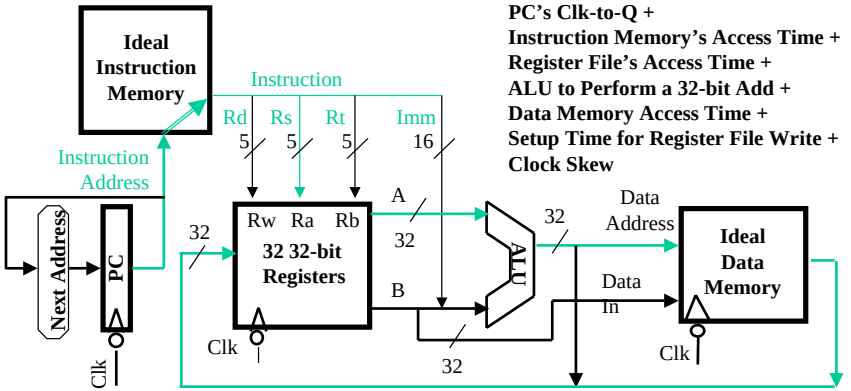
Putting it All Together: A Single Cycle Datapath



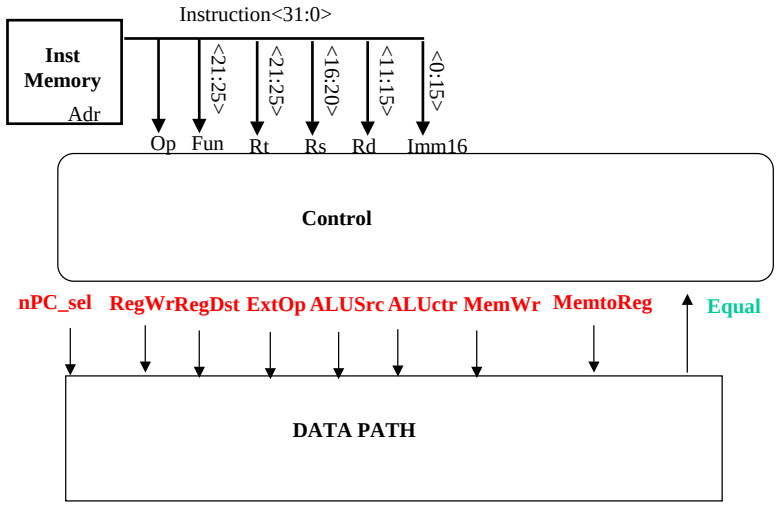
An Abstract View of the Critical Path

- **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."**

Critical Path (Load Operation) =
PC's Clk-to-Q +
Instruction Memory's Access Time +
Register File's Access Time +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Setup Time for Register File Write +
Clock Skew



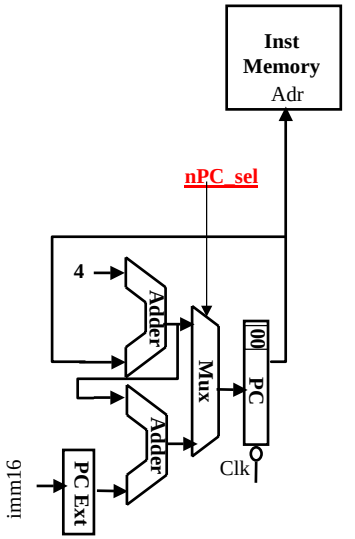
Step 4: Given Datapath: RTL → Control



Meaning of the Control Signals (see 3 slides back)

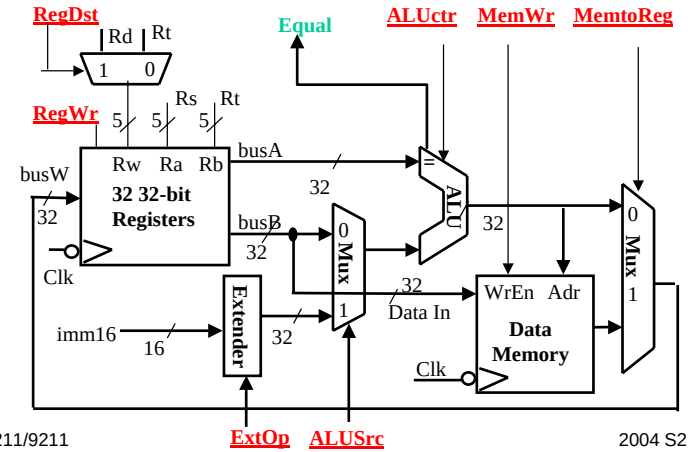
Rs, Rt, Rd and Imed16 hardwired into datapath

nPC_sel: 0 ⇒ PC ← PC + 4; 1 ⇒ PC ← PC + 4 + SignExt(Im16) || 00



Meaning of the Control Signals (see 4 slides back)

- **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

inst	Register Transfer	
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 = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__	
ORi	R[rt] ← R[rs] + zero_ext(Imm16);	PC ← PC + 4
	ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__	
LOAD	R[rt] ← MEM[R[rs] + sign_ext(Imm16)];	PC ← PC + 4
	ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__	
STORE	MEM[R[rs] + sign_ext(Imm16)] ← R[rs];	PC ← PC + 4
	ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =__	
BEQ	if (R[rs] == R[rt]) then PC ← PC + sign_ext(Imm16)] || 00 else PC ← PC + 4	
	ALUsrc = ___, Extop = __, ALUctr = ___, RegDst = ___, RegWr(?), MemtoReg(?), MemWr(?), nPC_sel =	

Control Signals

inst	Register Transfer	
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"	
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 == "000000") then "regB"
else "immed"

ALUctr ⟸ if (OP == "000000") then funct
elseif (OP == ORi) then "OR"
elseif (OP == BEQ) then "sub"
else "add"

ExtOp ⟸ _____

MemWr ⟸ _____

MemtoReg ⟸ _____

RegWr ⟸ _____

RegDst ⟸ _____

Step 5: Logic for each control signal

nPC_sel ⟸ if (OP == BEQ) then EQUAL else 0

ALUsrc ⟸ if (OP == "000000") then "regB"
else "immed"

ALUctr ⟸ if (OP == "000000") then funct
elseif (OP == ORi) then "OR"
elseif (OP == BEQ) then "sub"
else "add"

ExtOp ⟸ if (OP == ORi) then "zero" else "sign"

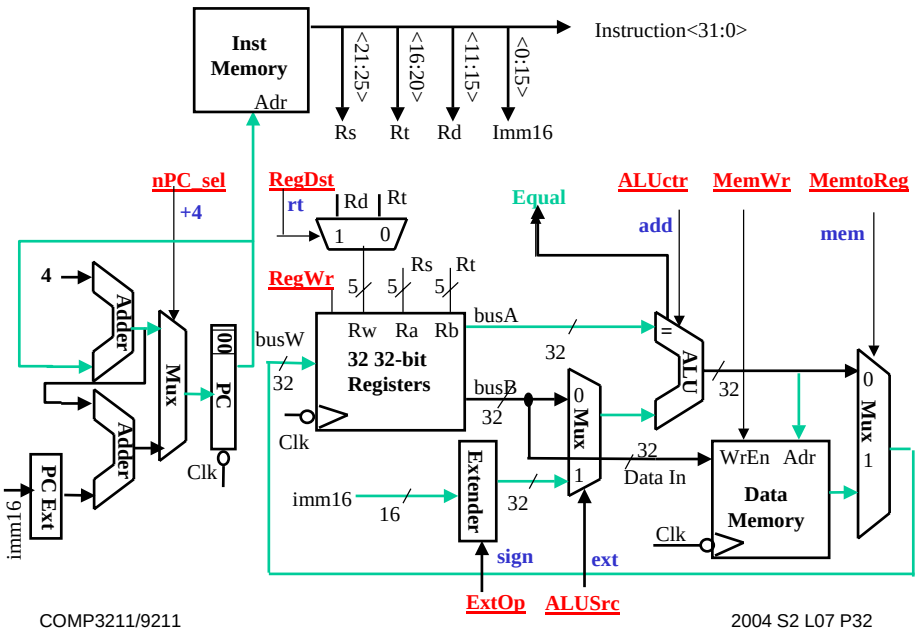
MemWr ⟸ (OP == Store)

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



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 affect the register transfer.**
  5. **Assemble the control logic**
- **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**