Single Cycle Datapath Design

Lecturer: Dr. Hui Annie Guo
huig@cse.unsw.edu.au
K17-501F (ext. 57136)

Overview

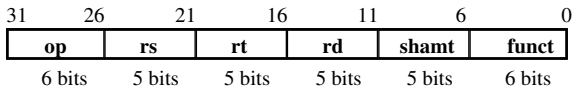
- Steps of processor design
- Single-cycle datapath

Typical Steps of Processor Design

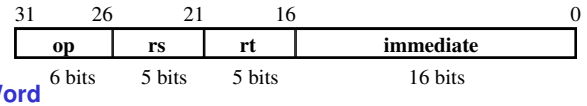
1. Analyse instruction set ⇒ datapath requirements
   - the meaning of each instruction is given by the register transfers
   - datapath must include storage element for ISA registers
   - datapath must support each register transfer
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting the requirements
4. Analyse implementation of each instruction to determine setting of control points that affect the register transfer.
5. Assemble the control logic

The MIPS-lite Subset

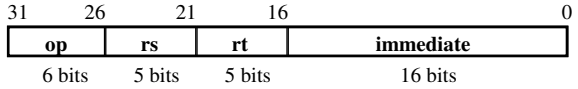
- We demonstrate processor design for the following instructions
  - ADD and SUB
    - addU rd, rs, rt
    - subU rd, rs, rt



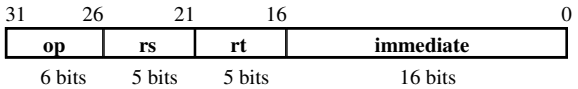
  - OR Immediate:
    - ori rt, rs, imm16



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



  - BRANCH:
    - beq rs, rt, imm16



Step 1a: RTL Specifications

- **Register Level Language (RTL) used to describe the execution of instructions**
  - **All instructions start by fetching the instruction**

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

    op | rs | rt | Imm16
  - **Then different operations**

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 1b: Requirements of the Instruction Set

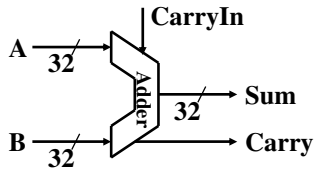
- **Memory**
  - **instruction & data**
- **Registers (let's say 32 x 32)**
  - **read RS**
  - **read RT**
  - **Write RT or RD**
- **PC**
- **Extender**
- **Add/Sub**
  - **register +/- register**
  - **Register +/- extended immediate**
- **Add**
  - **PC + 4**
  - **PC + 4 + extended immediate**

Step 2: Components of the Datapath

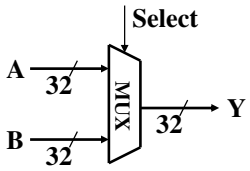
- **Combinational Elements**
- **Storage Elements**
  - **Clocking methodology**

Combinational Logic Elements (Building Blocks)

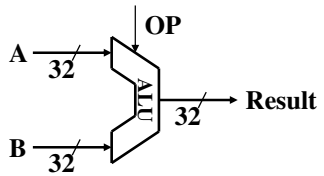
- **Adder**



- **MUX**

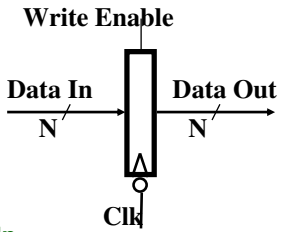


- **ALU**



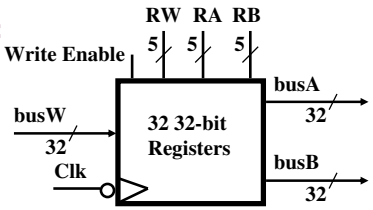
Storage Element: Register (Building Block)

- **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



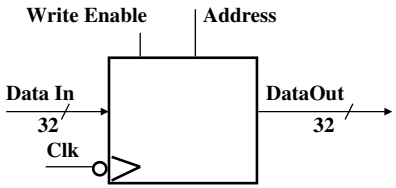
Storage Element: Register File

- **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."

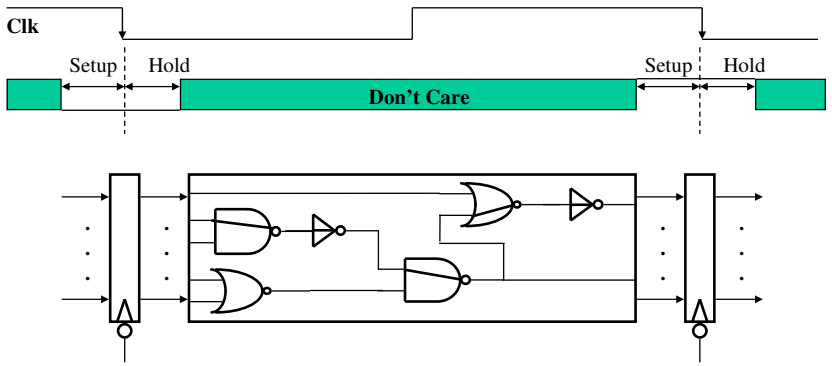


Storage Element: Memory

- **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.

We therefore need to ensure that the following 2 conditions are met:

1. Cycle Time ≥ CLK-to-Q + Longest Delay Path + Setup + Clock Skew
  - Clock skew = difference in arrival time of clock edges

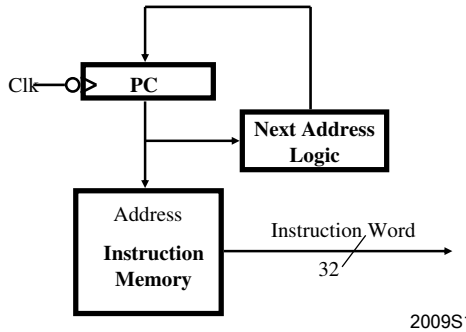
2. (CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time

Step 3

- **Register Transfer Requirements → Datapath Assembly**
- **Instruction Fetch**
- **Read Operands and Execute Operation**

3a: Overview of the Instruction Fetch Unit

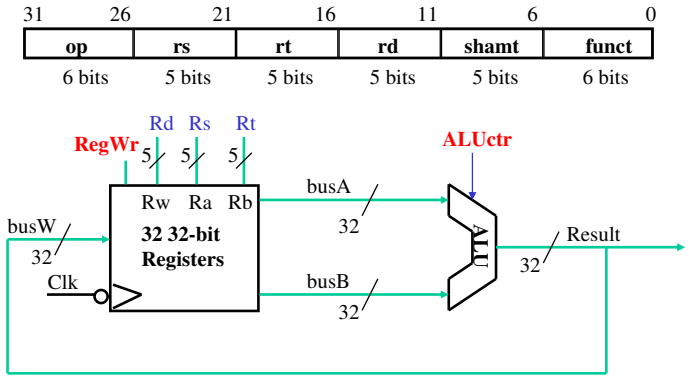
- **Common RTL operations:**
  - **At start of cycle, fetch the instruction: mem[PC]**
  - **At end of cycle, update the program counter:**
    - **Sequential Code: PC ← PC + 4**
    - **Branch and Jump: PC ← "something else"**



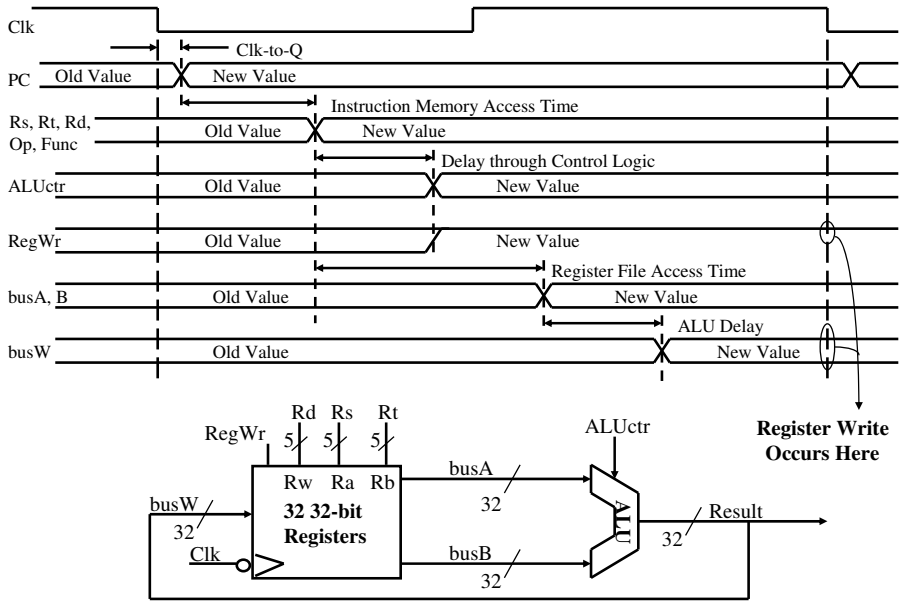
3b: Add & Subtract

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**

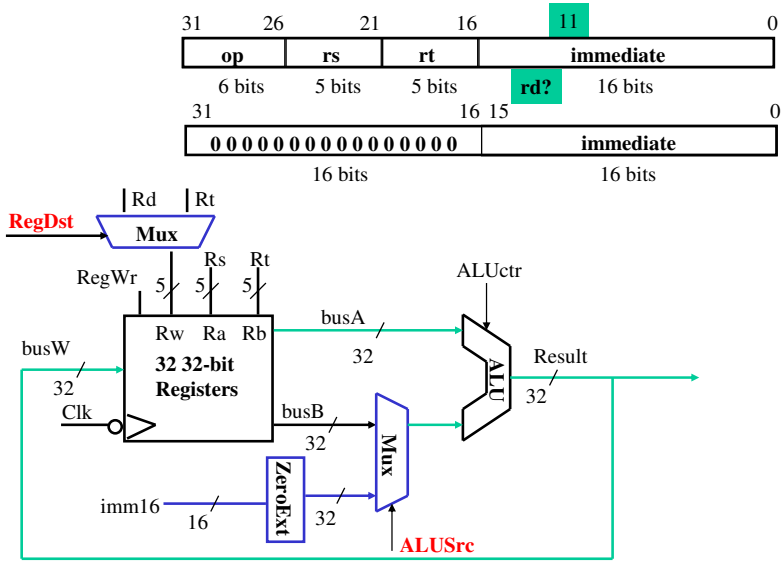


Register-Register Timing



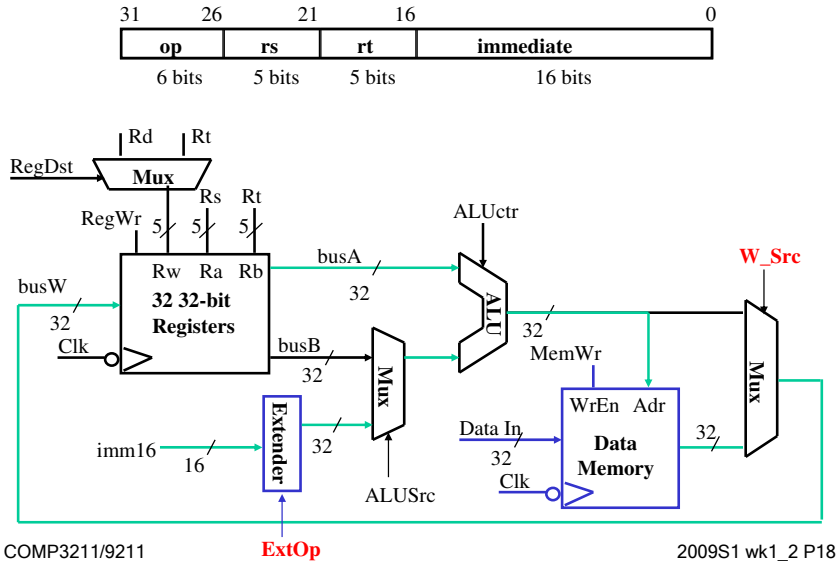
3c: Logical Operations with Immediate

R[rt] ← R[rs] op ZeroExt[imm16]



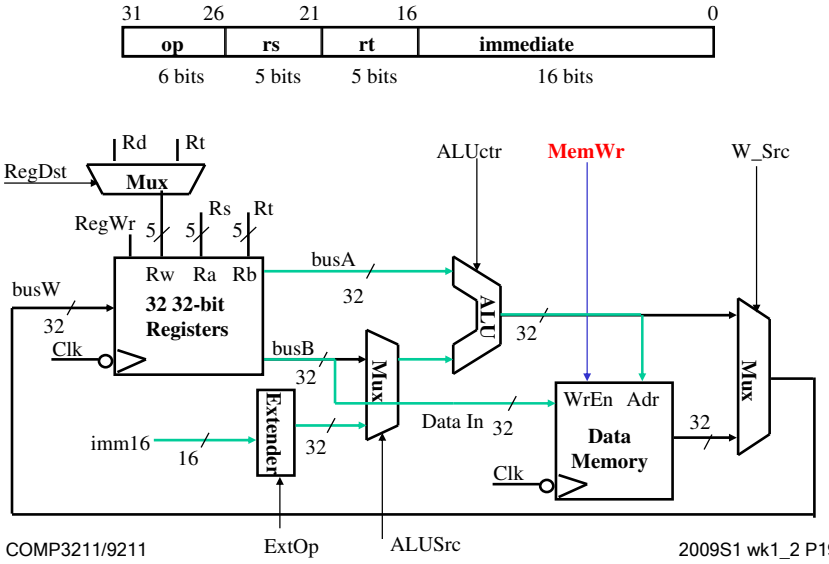
3d: Load Operations

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

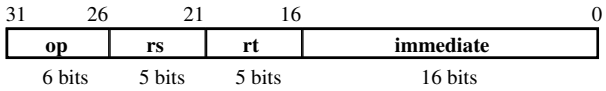


3e: Store Operations

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 1) Calculate the next instruction's address

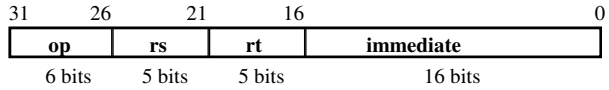
  PC ← PC + 4 + (SignExt(imm16) x 4)

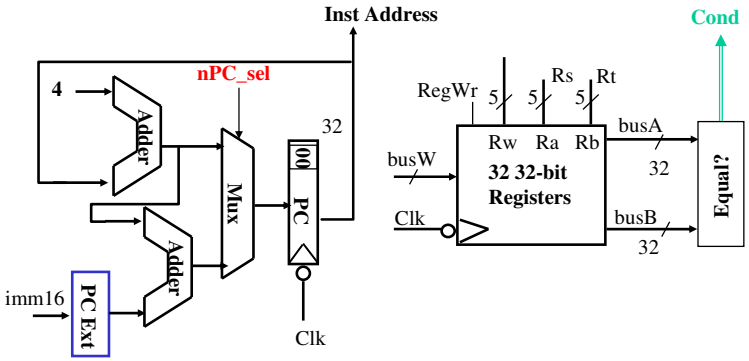
  else

  PC ← PC + 4

Datapath for Branch Operations

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





Putting it All Together: A Single Cycle Datapath