CS61C – Machine Structures

Lecture 27 - CPU Design: Designing a Single-cycle CPU, part II

3/23/2006

John Wawrzynek

(www.cs.berkeley.edu/~johnw)

www-inst.eecs.berkeley.edu/~cs61c/

How to Design a Processor: step-by-step

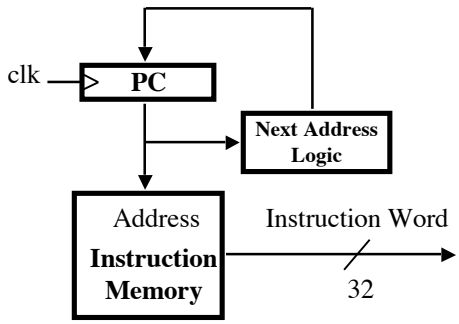
1. Analyze instruction set architecture (ISA) => datapath requirements
   - meaning of each instruction is given by the *register transfers*
   - datapath must include storage element for ISA registers
   - datapath must support each register transfer
2. Select set of datapath components and establish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

Step 3: Assemble DataPath meeting requirements

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

3a: Overview of the Instruction Fetch Unit

- Common to all instructions:
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC ← PC + 4
    - Branch and Jump: PC ← "something else"

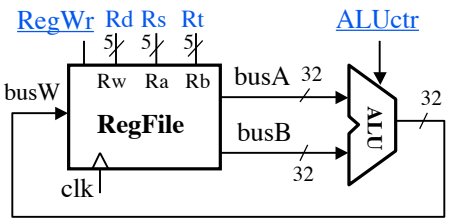


3b: Add & Subtract

- R[rd] ← R[rs] op R[rt] Ex.: `addU rd,rs,rt`
  - Ra, Rb, and Rw come from instruction's Rs, Rt, and Rd fields

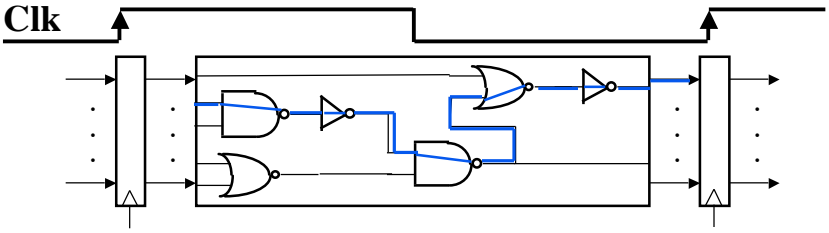
31–26	25–21	20–16	15–11	10–6	5–0
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

  - ALUctr and RegWr: control logic after decoding the instruction



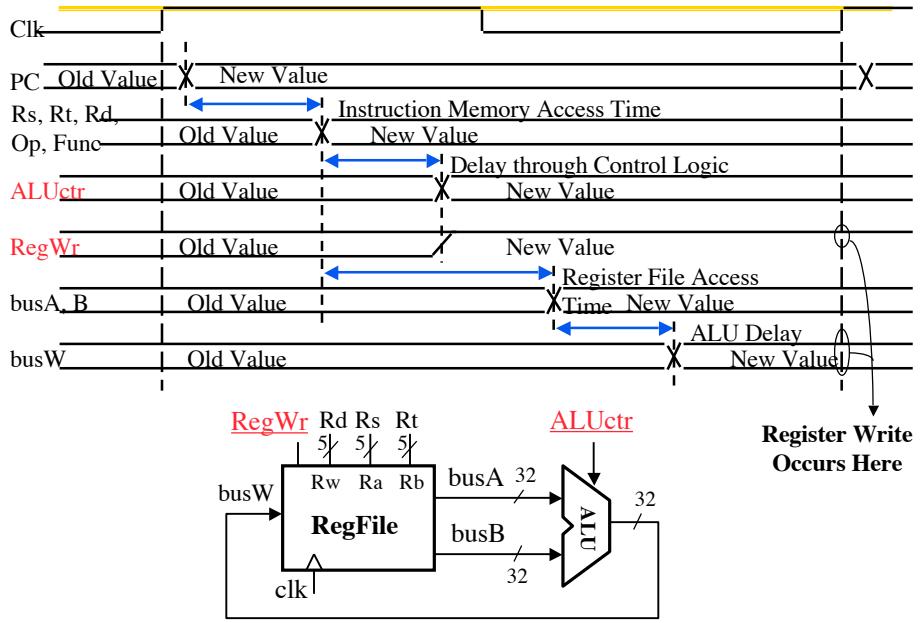
Already defined the register file & ALU

Clocking Methodology



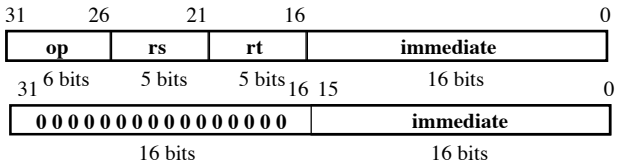
- Storage elements clocked by same edge
- Being physical devices, flip-flops (FF) and combinational logic have some delays
  - Gates: delay from input change to output change
  - Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF (set-up time), and we have the usual clock-to-Q delay
- "Critical path" (longest path through logic) determines length of clock period

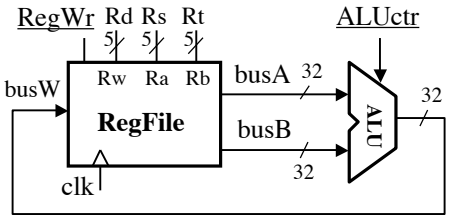
Register-Register Timing: One complete cycle



3c: Logical Operations with Immediate

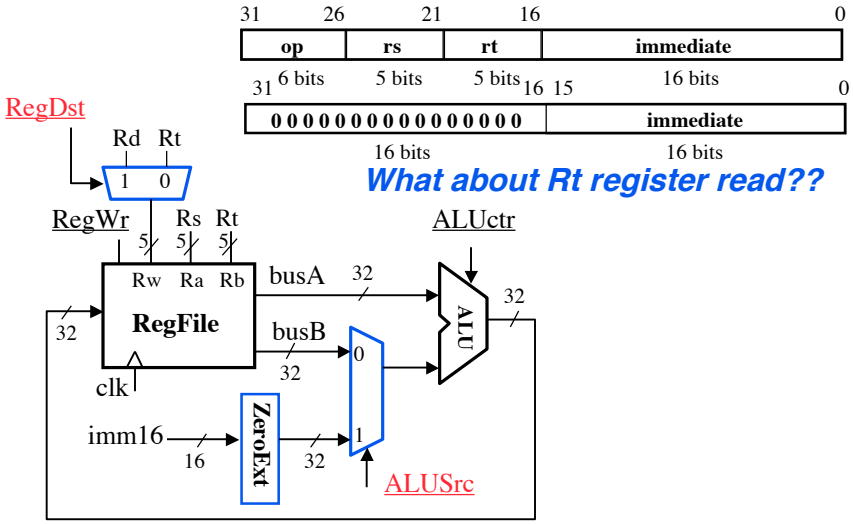
° R[rt] = R[rs] op ZeroExt[imm16]]





3c: Logical Operations with Immediate

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

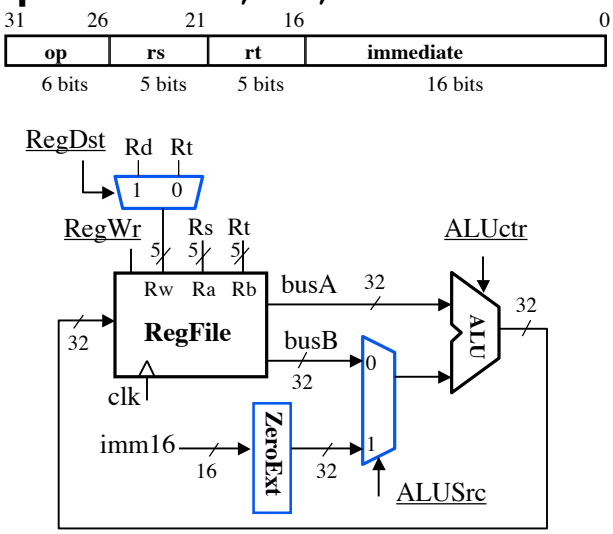


- Already defined 32-bit MUX; Zero Ext?

3d: Load Operations

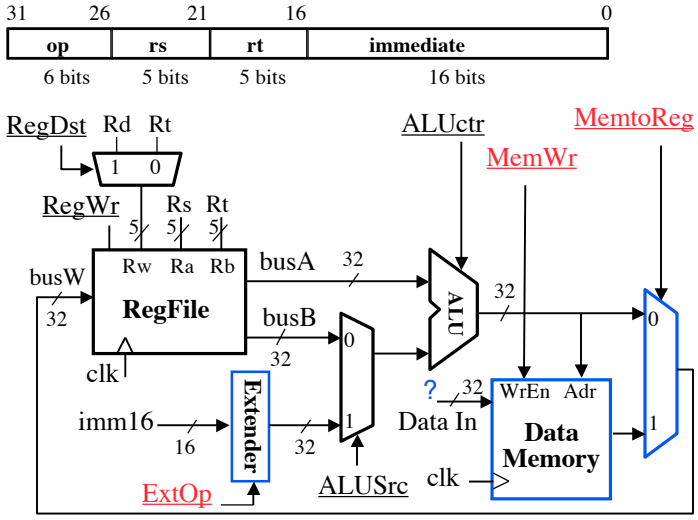
- R[rt] = Mem[R[rs] + SignExt[imm16]]

Example: `lw rt,rs,imm16`



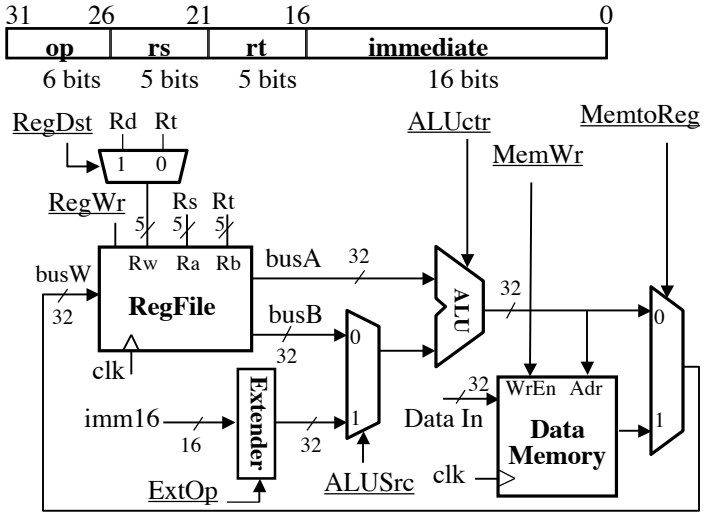
3d: Load Operations

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



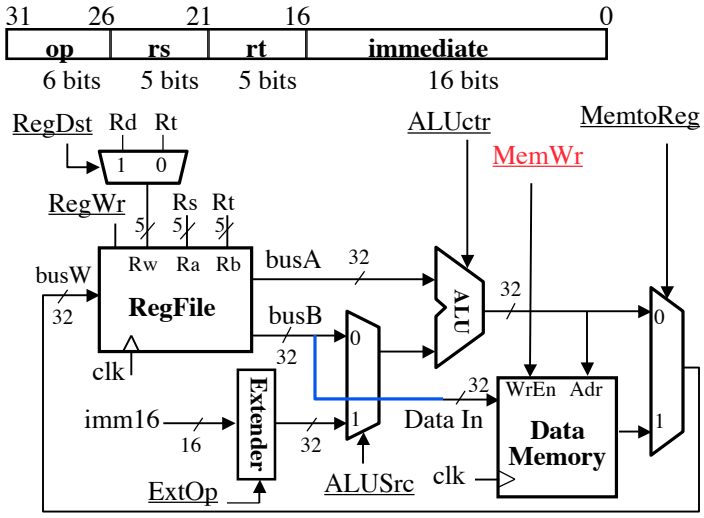
3e: Store Operations

- Mem[R[rs] + SignExt[imm16]] = R[rt]
  Ex.: `sw rt, rs, imm16`

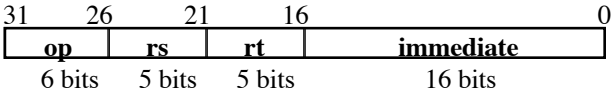


3e: Store Operations

° Mem[R[rs] + SignExt[imm16]] = R[rt]
Ex.: `sw rt, rs, imm16`



3f: The Branch Instruction



`beq rs, rt, imm16`

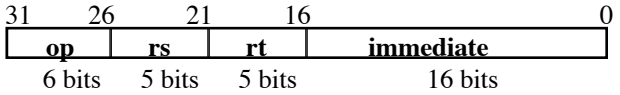
- mem[PC] Fetch the instruction from memory
- Equal = R[rs] == R[rt] Calculate branch condition
- if (Equal) Calculate the next instruction's address
  - PC = PC + 4 + (SignExt(imm16) x 4)

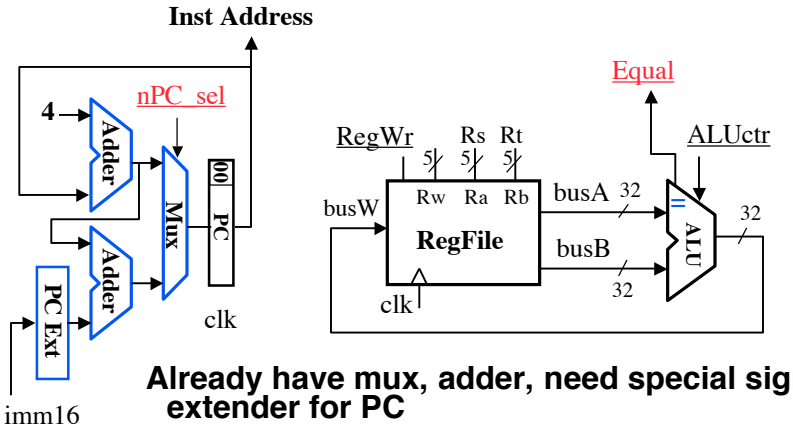
else
  - PC = PC + 4

Datapath for Branch Operations

- beq rs, rt, imm16

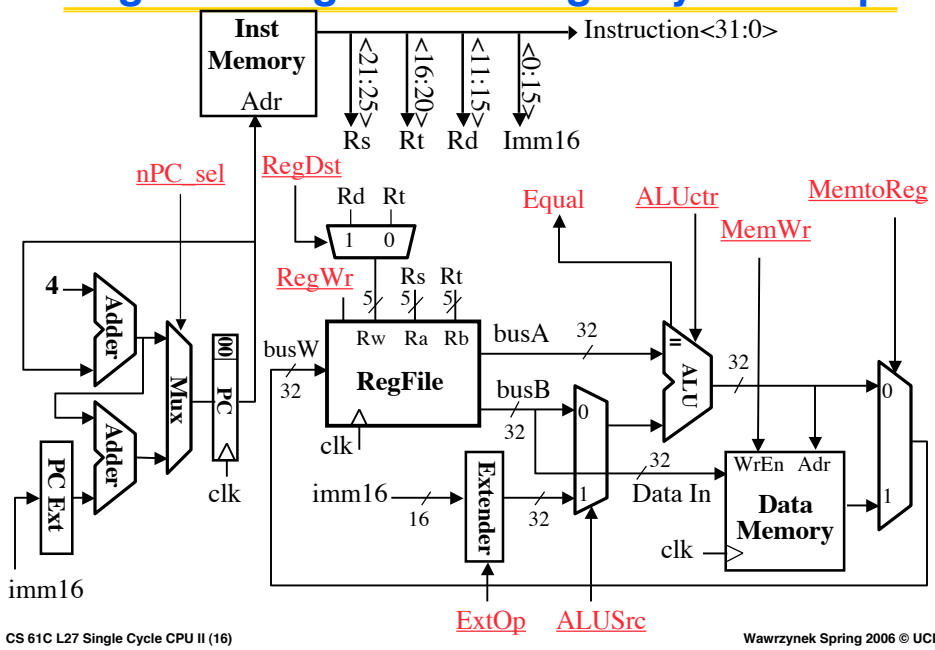
Datapath generates condition (equal)



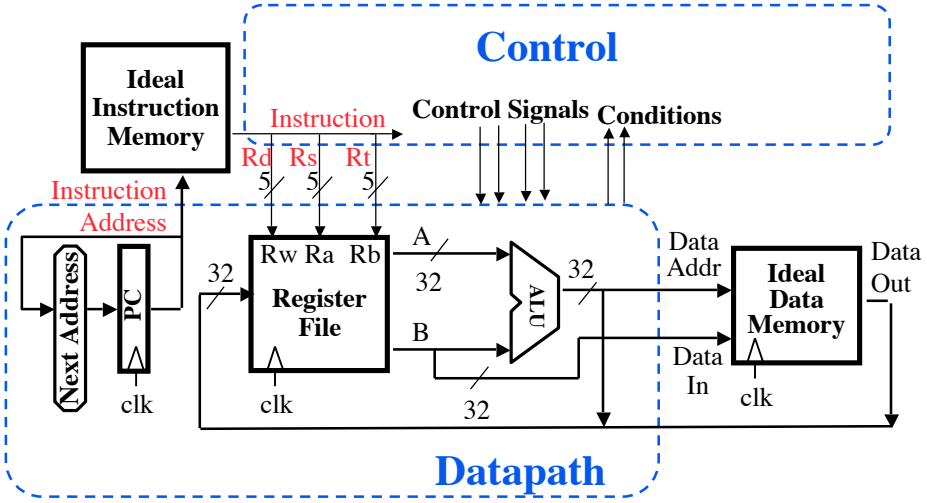


Already have mux, adder, need special sign extender for PC

Putting it All Together:A Single Cycle Datapath



An Abstract View of the Implementation



Summary: Single cycle datapath

- 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
- Next time!