ECE4680
Computer Organization and Architecture
Designing a Single Cycle Datapath

Processor Design: How to Implement MIPS

Simplicity favors regularity

The Big Picture: Where are We Now?

° The Five Classic Components of a Computer

Processor		
Control	Memory	Input
Datapath		Output

° Today's Topic: Datapath Design
- What is data?
- What is datapath?

The Big Picture: The Performance Perspective

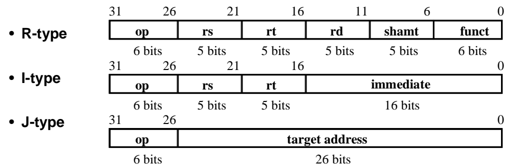
° Performance of a machine was determined by:
- Instruction count
- Clock cycle time
- Clock cycles per instruction

° Processor design (datapath and control) will determine:
- Clock cycle time
- Clock cycles per instruction

° In the next two lectures:
- Single cycle processor:
  - Advantage: One clock cycle per instruction
  - Disadvantage: long cycle time

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

The MIPS Subset

° ADD and subtract
- add rd, rs, rt
- sub rd, rs, rt

31 op 26	21 rs	16 rt	11 rd	6 shamt	funct 0
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

° OR Immediate:
- ori rt, rs, imm16

31 op 26	21 rs	16 rt	immediate 0
6 bits	5 bits	5 bits	16 bits

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

° BRANCH:
- beq rs, rt, imm16

° JUMP:
- j target

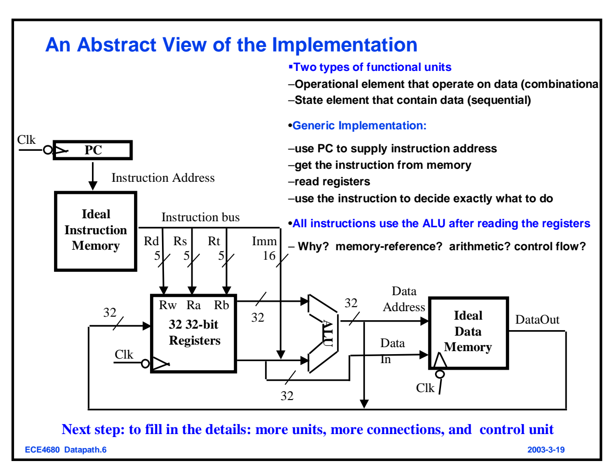
31 op 26	target address 0
6 bits	26 bits

An Abstract View of the Implementation

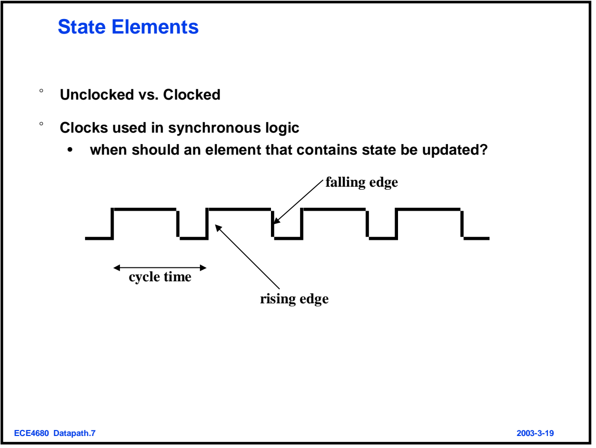
•Two types of functional units
–Operational element that operate on data (combinationa
–State element that contain data (sequential)

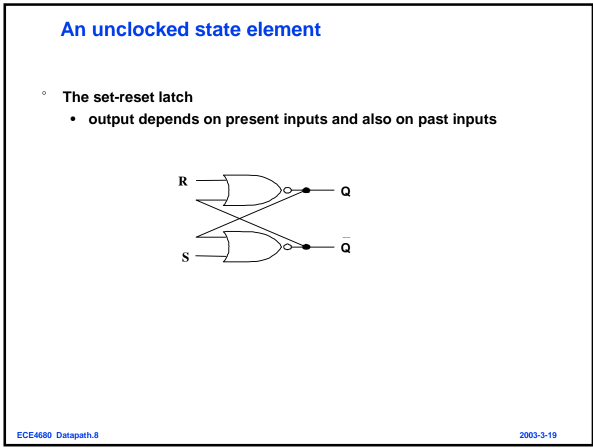
•Generic Implementation:
–use PC to supply instruction address
–get the instruction from memory
–read registers
–use the instruction to decide exactly what to do

•All instructions use the ALU after reading the registers
– Why? memory-reference? arithmetic? control flow?



Next step: to fill in the details: more units, more connections, and control unit



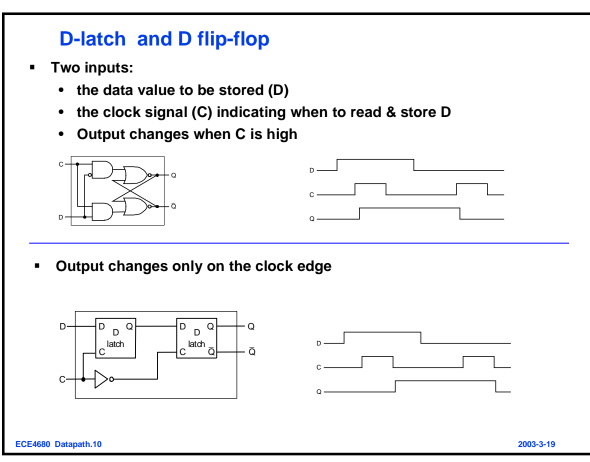


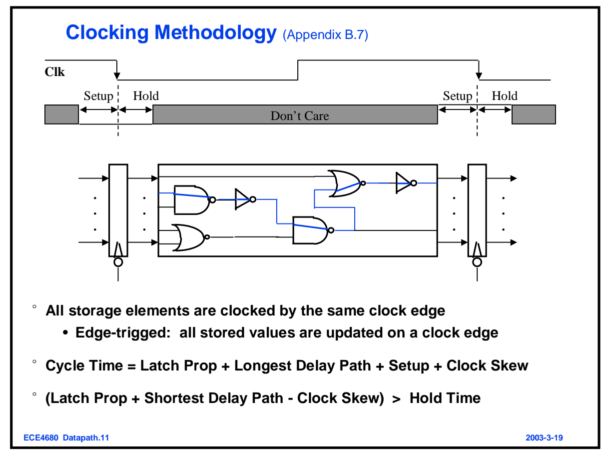
Latches and Flip-flops

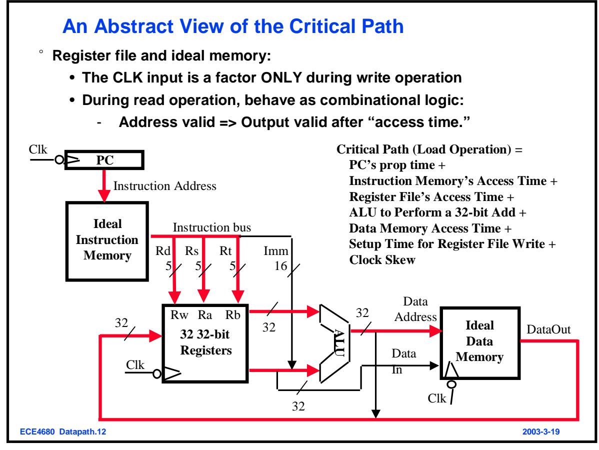
- Output is equal to the stored value inside the element
  (don't need to ask for permission to look at the value)
- Change of state (value) is based on the clock
- Latches: whenever the inputs change, and the clock is asserted
- Flip-flop: state changes only on a clock edge
  (edge-triggered methodology)

"logically true",
— could mean electrically low

A clocking methodology defines when signals can be read and written
— wouldn't want to read a signal at the same time it was being written





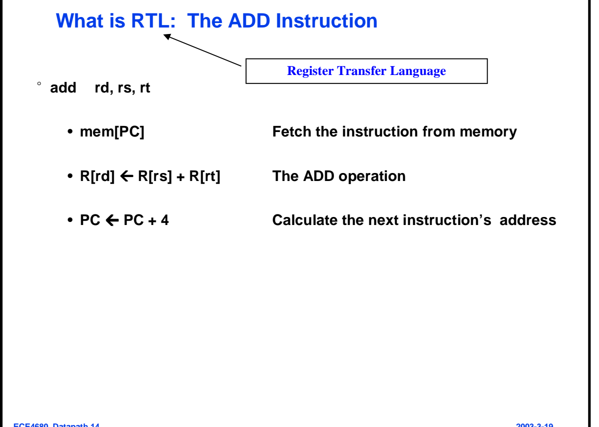


The Steps of Designing a Processor

- Instruction Set Architecture => Register Transfer Language
- Register Transfer Language (RTL) =>
  - Datapath components
  - Datapath interconnect
- Datapath components => Control signals
- Control signals => Control logic

Element < component

What is RTL: The ADD Instruction



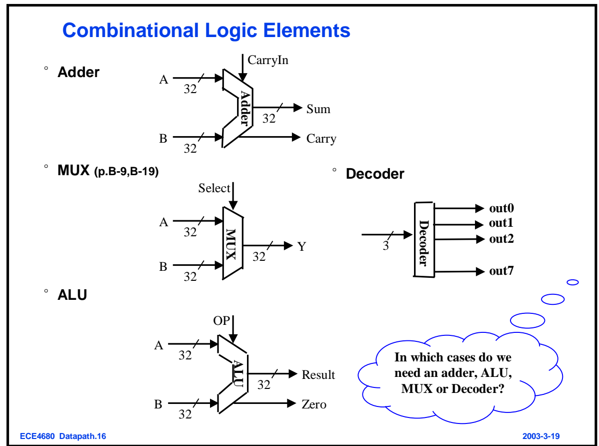
- add rd, rs, rt
  - mem[PC] — Fetch the instruction from memory
  - R[rd] ← R[rs] + R[rt] — The ADD operation
  - PC ← PC + 4 — Calculate the next instruction's address

What is RTL: The Load Instruction

- lw rt, rs, imm16
  - mem[PC] — Fetch the instruction from memory
  - Addr ← R[rs] + SignExt(imm16) — Calculate the memory address
  - R[rt] ← Mem[Addr] — Load the data into the register
  - PC ← PC + 4 — Calculate the next instruction's address

Combinational Logic Elements

- Adder
- MUX (p.B-9,B-19)
- Decoder
- ALU



Storage Element: Register (p.B22-B25)

- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
  - Write Enable:
    - 0: Data Out will not change
    - 1: Data Out will become Data In
  - Array of logical elements(see register file on next 2 slides)

Write Enable
Data In
N
Data Out
N
Clk

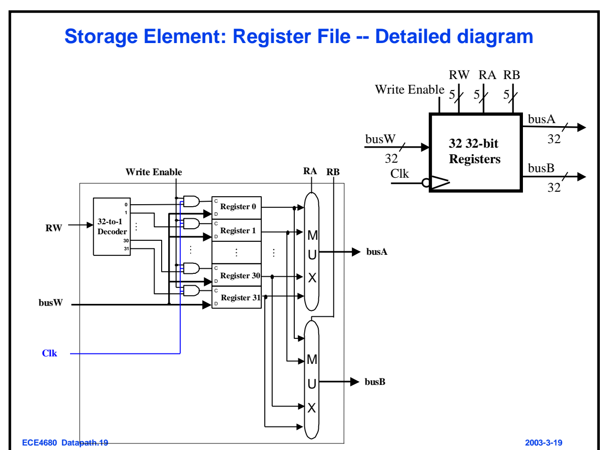
The content is updated at the clock tick ONLY if the Write Enable signal is set to 1.

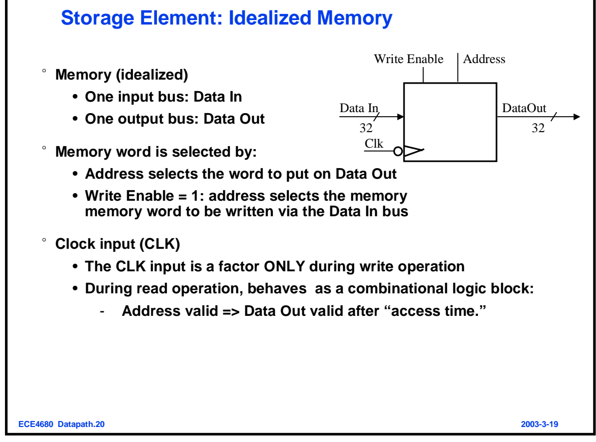
Storage Element: Register File

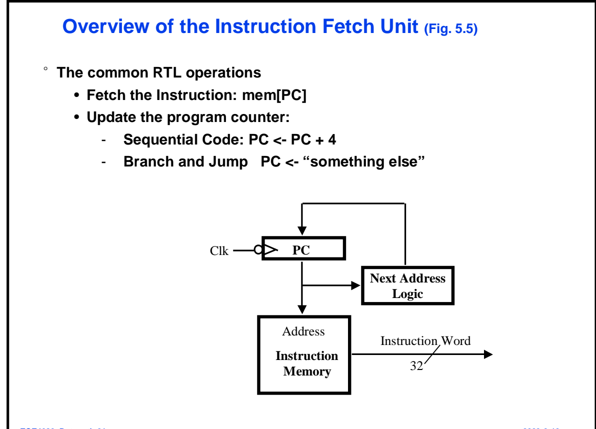
- Register File consists of 32 registers:
  - Two 32-bit output busses:
    busA and busB
  - One 32-bit input bus: busW
- Register is selected by:
  - RA selects the register to put on busA
  - RB selects the register to put on busB
  - RW selects the register to be written via busW when Write Enable is 1

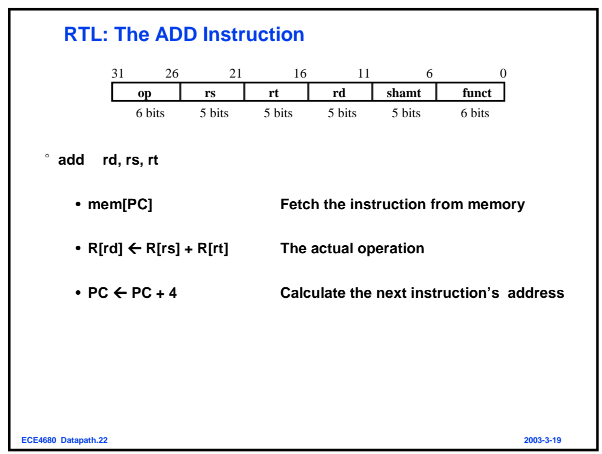
RW RA RB
Write Enable 5 5 5
busA 32
busW 32
32 32-bit Registers
busB 32
Clk

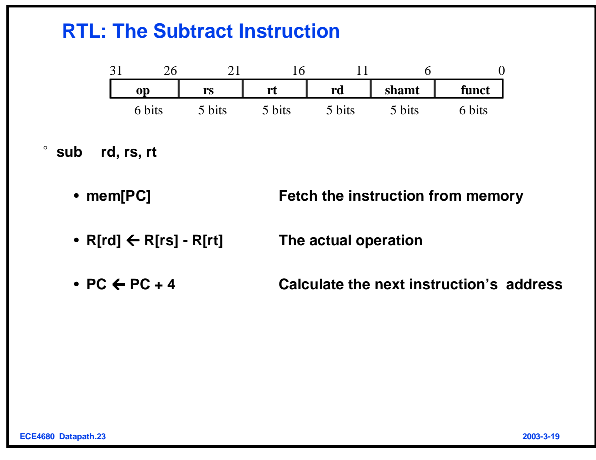
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid => busA or busB valid after "access time."

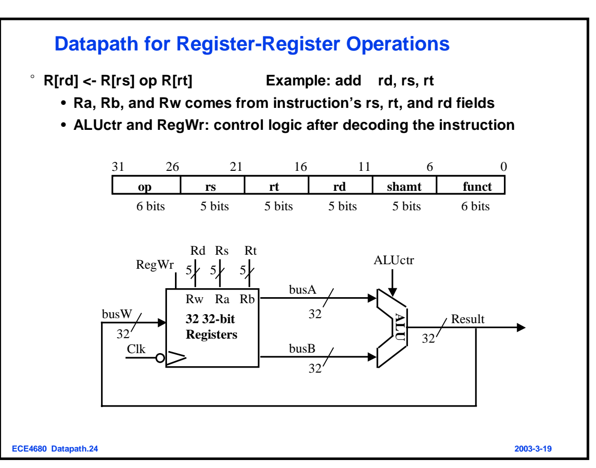

Storage Element: Register File -- Detailed diagram
RW RA RB
Write Enable
5 5 5
busA
32
busW
32 32-bit
Registers
32
Clk
busB
32
Write Enable
RA RB
RW
32-to-1
Decoder
Register 0
Register 1
Register 30
Register 31
M
U
X
busA
busW
Clk
M
U
X
busB
ECE4680 Datapath.19
2003-3-19

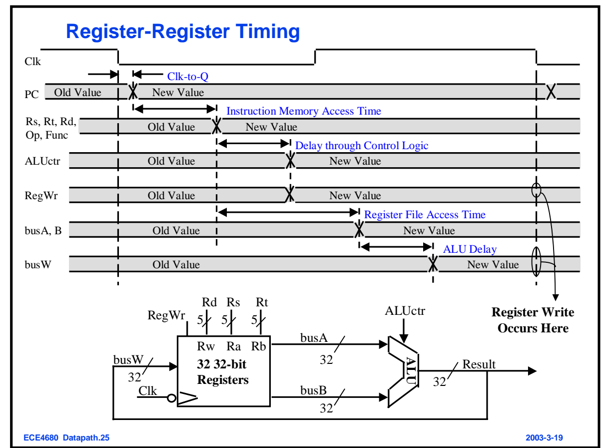

Storage Element: Idealized Memory
Memory (idealized)
• One input bus: Data In
• One output bus: Data Out
Memory word is selected by:
• Address selects the word to put on Data Out
• Write Enable = 1: address selects the memory memory word to be written via the Data In bus
Clock input (CLK)
• The CLK input is a factor ONLY during write operation
• During read operation, behaves as a combinational logic block:
- Address valid => Data Out valid after "access time."
Write Enable
Address
Data In
32
Clk
DataOut
32
ECE4680 Datapath.20
2003-3-19

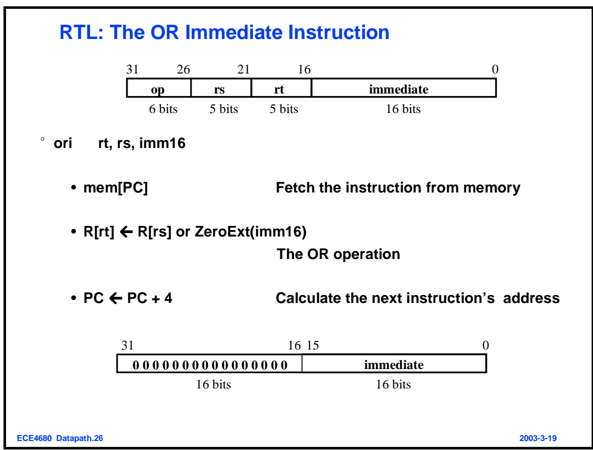

Overview of the Instruction Fetch Unit (Fig. 5.5)
The common RTL operations
• Fetch the Instruction: mem[PC]
• Update the program counter:
- Sequential Code: PC <- PC + 4
- Branch and Jump PC <- "something else"
Clk
PC
Next Address
Logic
Address
Instruction
Memory
Instruction Word
32
ECE4680 Datapath.21
2003-3-19

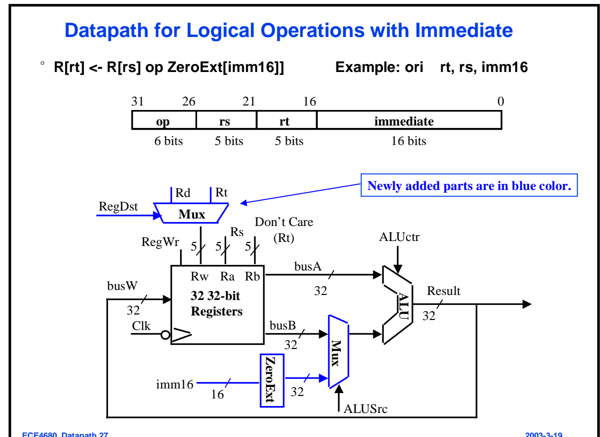

RTL: The ADD Instruction
31 26 21 16 11 6 0
op rs rt rd shamt funct
6 bits 5 bits 5 bits 5 bits 5 bits 6 bits
add rd, rs, rt
• mem[PC] Fetch the instruction from memory
• R[rd] ← R[rs] + R[rt] The actual operation
• PC ← PC + 4 Calculate the next instruction's address
ECE4680 Datapath.22
2003-3-19

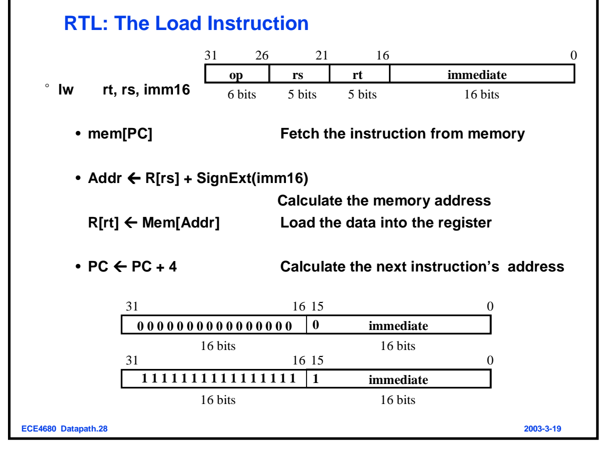

RTL: The Subtract Instruction
31 26 21 16 11 6 0
op rs rt rd shamt funct
6 bits 5 bits 5 bits 5 bits 5 bits 6 bits
sub rd, rs, rt
• mem[PC] Fetch the instruction from memory
• R[rd] ← R[rs] - R[rt] The actual operation
• PC ← PC + 4 Calculate the next instruction's address
ECE4680 Datapath.23
2003-3-19

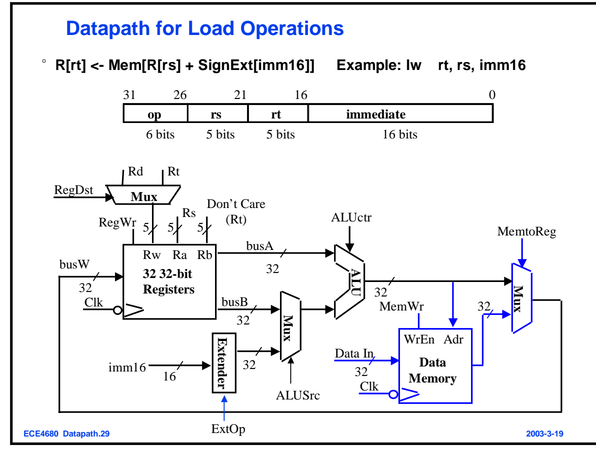

Datapath for Register-Register Operations
R[rd] <- R[rs] op R[rt] Example: add rd, rs, rt
• Ra, Rb, and Rw comes from instruction's rs, rt, and rd fields
• ALUctr and RegWr: control logic after decoding the instruction
31 26 21 16 11 6 0
op rs rt rd shamt funct
6 bits 5 bits 5 bits 5 bits 5 bits 6 bits
RegWr
Rd Rs Rt
5 5 5
ALUctr
busA
32
Rw Ra Rb
32 32-bit
Registers
busW
32
Clk
busB
32
ALU
Result
32
ECE4680 Datapath.24
2003-3-19

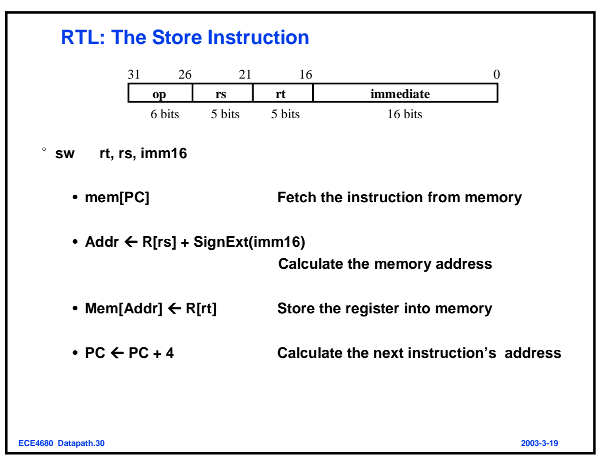

Register-Register Timing
Clk
Clk-to-Q
PC
Old Value
New Value
Instruction Memory Access Time
Rs, Rt, Rd,
Op, Func
Delay through Control Logic
ALUctr
RegWr
Register File Access Time
busA, B
ALU Delay
busW
Register Write
Occurs Here
Rd Rs Rt
RegWr
Rw Ra Rb
32 32-bit
Registers
busA
busB
busW
Clk
ALUctr
ALU
Result
ECE4680 Datapath.25
2003-3-19


RTL: The OR Immediate Instruction
31 26 21 16 0
op rs rt immediate
6 bits 5 bits 5 bits 16 bits
ori rt, rs, imm16
• mem[PC] Fetch the instruction from memory
• R[rt] ← R[rs] or ZeroExt(imm16)
The OR operation
• PC ← PC + 4 Calculate the next instruction's address
31 16 15 0
0000000000000000 immediate
16 bits 16 bits
ECE4680 Datapath.26
2003-3-19


Datapath for Logical Operations with Immediate
R[rt] <- R[rs] op ZeroExt[imm16]] Example: ori rt, rs, imm16
31 26 21 16 0
op rs rt immediate
6 bits 5 bits 5 bits 16 bits
Newly added parts are in blue color.
RegDst
Mux
Rd Rt
Don't Care
(Rt)
Rs
RegWr
Rw Ra Rb
32 32-bit
Registers
busA
busB
busW
Clk
ALUctr
ALU
Result
ZeroExt
Mux
imm16
ALUSrc
ECE4680 Datapath.27
2003-3-19


RTL: The Load Instruction
31 26 21 16 0
op rs rt immediate
6 bits 5 bits 5 bits 16 bits
lw rt, rs, imm16
• mem[PC] Fetch the instruction from memory
• Addr ← R[rs] + SignExt(imm16)
Calculate the memory address
R[rt] ← Mem[Addr] Load the data into the register
• PC ← PC + 4 Calculate the next instruction's address
31 16 15 0
0000000000000000 0 immediate
16 bits 16 bits
1111111111111111 1 immediate
16 bits 16 bits
ECE4680 Datapath.28
2003-3-19

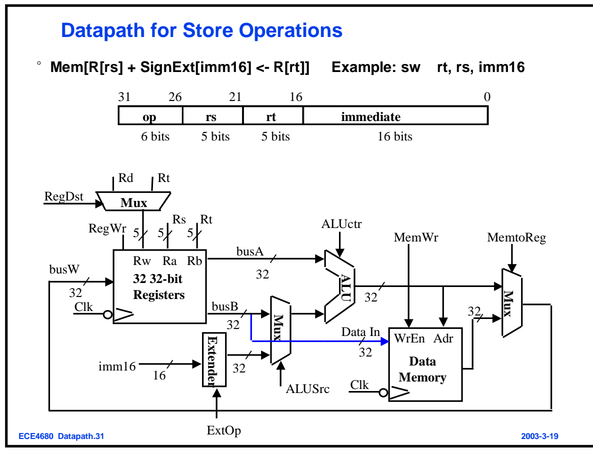

Datapath for Load Operations
R[rt] <- Mem[R[rs] + SignExt[imm16]] Example: lw rt, rs, imm16
31 26 21 16 0
op rs rt immediate
6 bits 5 bits 5 bits 16 bits
RegDst
Mux
Rd Rt
Don't Care
(Rt)
Rs
RegWr
Rw Ra Rb
32 32-bit
Registers
busA
busB
busW
Clk
ALUctr
ALU
MemtoReg
MemWr
Mux
Extender
imm16
ALUSrc
ExtOp
WrEn Adr
Data In
Data
Memory
ECE4680 Datapath.29
2003-3-19


RTL: The Store Instruction
31 26 21 16 0
op rs rt immediate
6 bits 5 bits 5 bits 16 bits
sw rt, rs, imm16
• mem[PC] Fetch the instruction from memory
• Addr ← R[rs] + SignExt(imm16)
Calculate the memory address
• Mem[Addr] ← R[rt] Store the register into memory
• PC ← PC + 4 Calculate the next instruction's address
ECE4680 Datapath.30
2003-3-19

Datapath for Store Operations

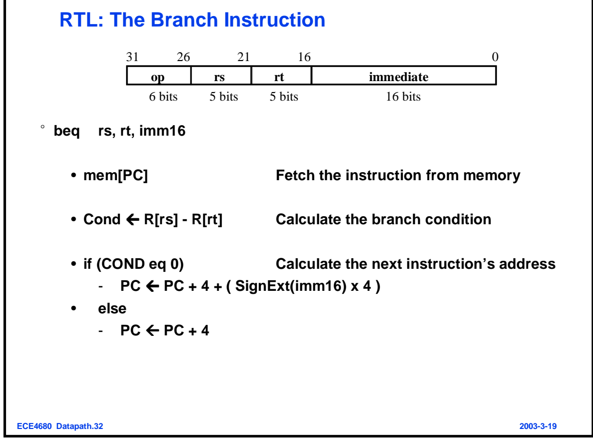
- Mem[R[rs] + SignExt[imm16] <- R[rt]]   Example: sw   rt, rs, imm16

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	



RTL: The Branch Instruction

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	

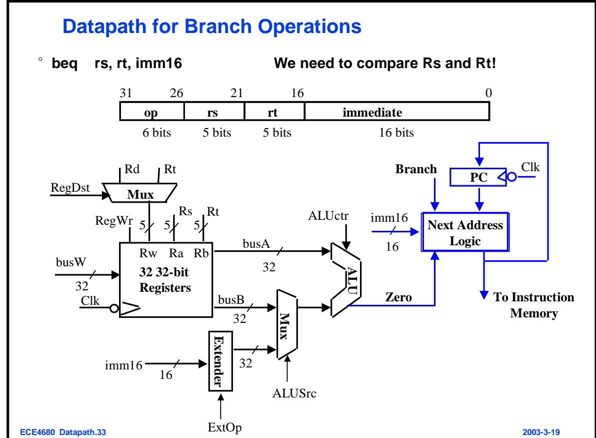


- beq   rs, rt, imm16
  - mem[PC]   Fetch the instruction from memory
  - Cond ← R[rs] - R[rt]   Calculate the branch condition
  - if (COND eq 0)   Calculate the next instruction's address
    - PC ← PC + 4 + (SignExt(imm16) x 4)
  - else
    - PC ← PC + 4

Datapath for Branch Operations

- beq   rs, rt, imm16   We need to compare Rs and Rt!

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	

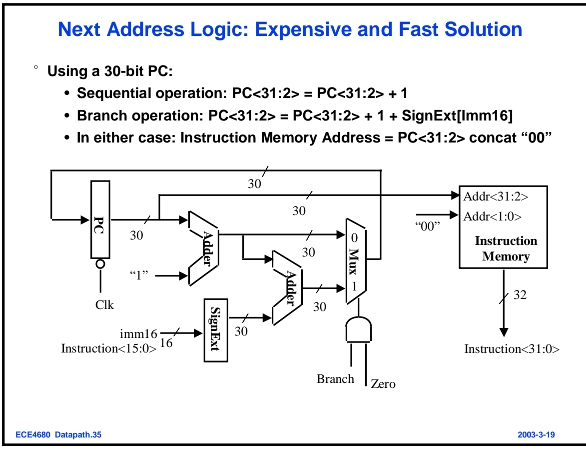


Binary Arithmetic for the Next Address

- In theory, the PC is a 32-bit byte address into the instruction memory:
  - Sequential operation: PC<31:0> = PC<31:0> + 4
  - Branch operation: PC<31:0> = PC<31:0> + 4 + SignExt[Imm16] * 4
- The magic number "4" always comes up because:
  - The 32-bit PC is a byte address
  - And all our instructions are 4 bytes (32 bits) long
- In other words:
  - The 2 LSBs of the 32-bit PC are always zeros
  - There is no reason to have hardware to keep the 2 LSBs
- In practice, we can simplify the hardware by using a 30-bit PC<31:2>:
  - Sequential operation: PC<31:2> = PC<31:2> + 1
  - Branch operation: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
  - In either case: Instruction Memory Address = PC<31:2> concat "00"

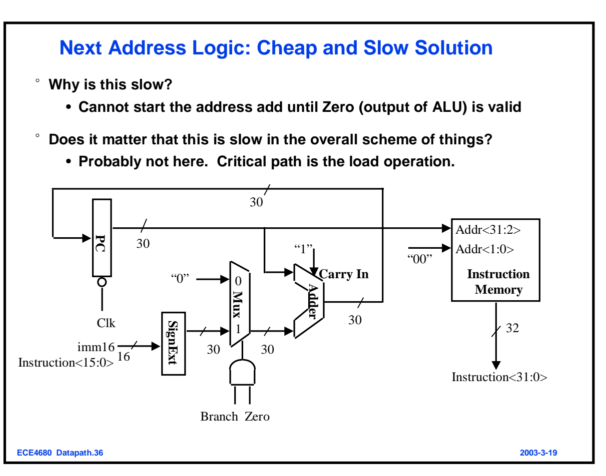
Next Address Logic: Expensive and Fast Solution

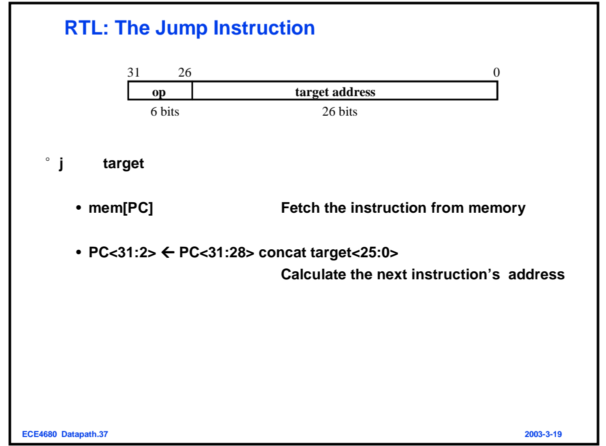
- Using a 30-bit PC:
  - Sequential operation: PC<31:2> = PC<31:2> + 1
  - Branch operation: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
  - In either case: Instruction Memory Address = PC<31:2> concat "00"

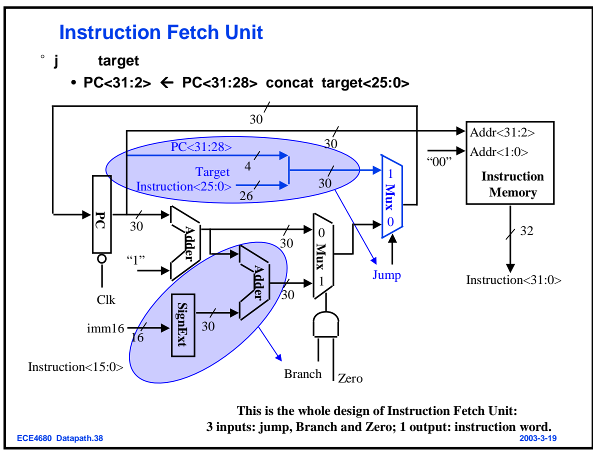


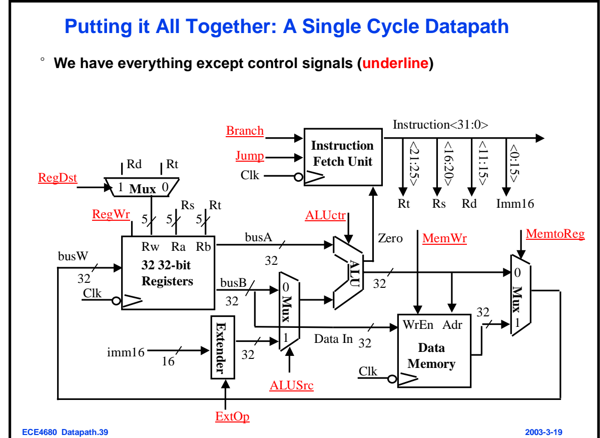
Next Address Logic: Cheap and Slow Solution

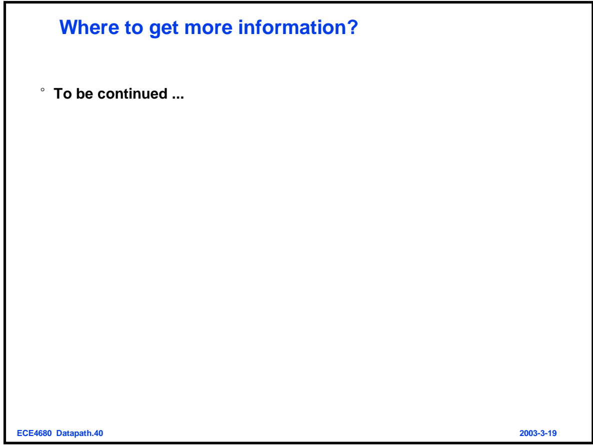
- Why is this slow?
  - Cannot start the address add until Zero (output of ALU) is valid
- Does it matter that this is slow in the overall scheme of things?
  - Probably not here. Critical path is the load operation.




RTL: The Jump Instruction
31
26
0
op
target address
6 bits
26 bits
° j target
• mem[PC]
Fetch the instruction from memory
• PC<31:2> ← PC<31:28> concat target<25:0>
Calculate the next instruction's address
ECE4680 Datapath.37
2003-3-19


Instruction Fetch Unit
° j target
• PC<31:2> ← PC<31:28> concat target<25:0>
PC<31:28>
Target
Instruction<25:0>
Addr<31:2>
Addr<1:0>
"00"
Instruction
Memory
Mux
Adder
"1"
Clk
imm16
SignExt
Instruction<15:0>
Jump
Branch
Zero
Instruction<31:0>
This is the whole design of Instruction Fetch Unit:
3 inputs: jump, Branch and Zero; 1 output: instruction word.
ECE4680 Datapath.38
2003-3-19


Putting it All Together: A Single Cycle Datapath
° We have everything except control signals (underline)
Branch
Jump
Clk
Instruction
Fetch Unit
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
Rt
Rs
Rd
Imm16
RegDst
Rd
Rt
Mux
RegWr
Rs
Rt
ALUctr
Zero
MemWr
MemtoReg
busA
busW
Rw Ra Rb
32 32-bit
Registers
busB
ALU
Mux
WrEn Adr
Data In
Data
Memory
imm16
Extender
ALUSrc
ExtOp
ECE4680 Datapath.39
2003-3-19


Where to get more information?
° To be continued ...
ECE4680 Datapath.40
2003-3-19