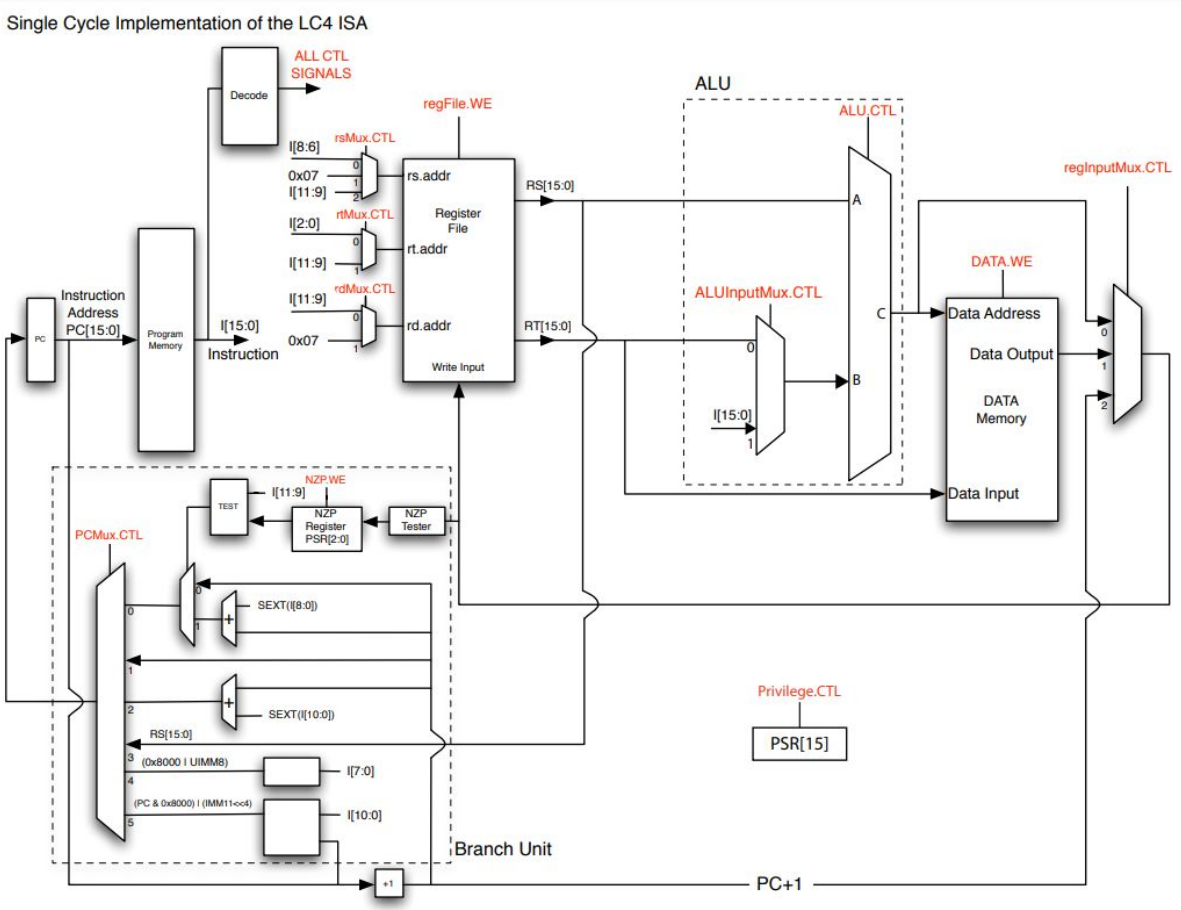
Datapath and Control Signals

CIS 2400 Recitation 5

Recitation Outline

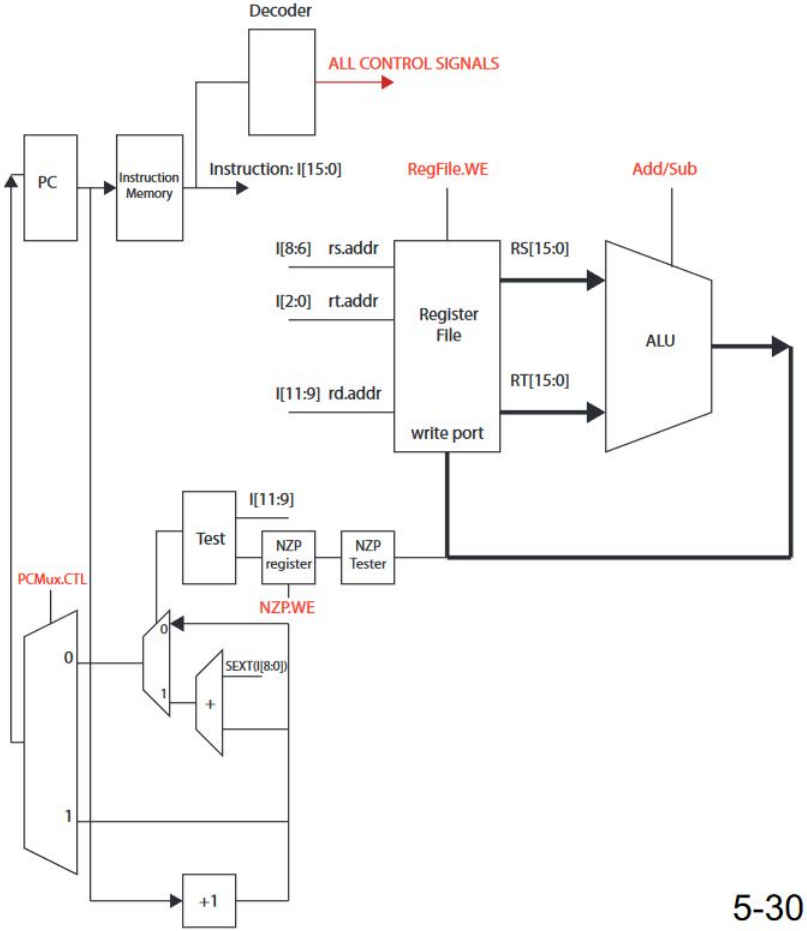
- Datapath & Control Signals
  - Register File
  - ALU
  - Memory
  - Branch Unit
- Practice Problems

Datapath & Control Signals


Single Cycle Implementation of the LC4 ISA
Decode
ALL CTL SIGNALS
regFile.WE
ALU
ALU.CTL
rsMux.CTL
I[8:6]
0x07
I[11:9]
rs.addr
Register File
rt.addr
rtMux.CTL
I[2:0]
I[11:9]
rdMux.CTL
I[11:9]
0x07
rd.addr
Write Input
RS[15:0]
RT[15:0]
regInputMux.CTL
ALUInputMux.CTL
I[15:0]
A
B
C
DATA.WE
Data Address
Data Output
DATA Memory
Data Input
PC
Instruction Address PC[15:0]
Program Memory
I[15:0]
Instruction
NZP.WE
TEST
I[11:9]
NZP Register PSR[2:0]
NZP Tester
PCMux.CTL
SEXT(I[8:0])
SEXT(I[10:0])
RS[15:0]
(0x8000 | UIMM8)
I[7:0]
(PC & 0x8000) | (IMM11<<4)
I[10:0]
+1
Branch Unit
PC+1
Privilege.CTL
PSR[15]

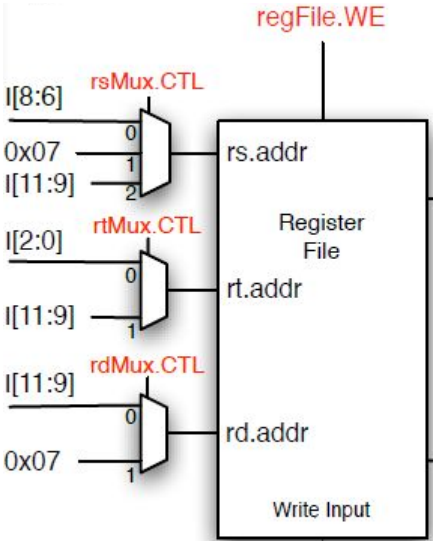
This is Datapath

- This is a lot
- What does it do?
  - Executes Code
  - Has many control signals to “control” execution
  - Executes code “one step at a time”
  - …



Datapath Components: Register File

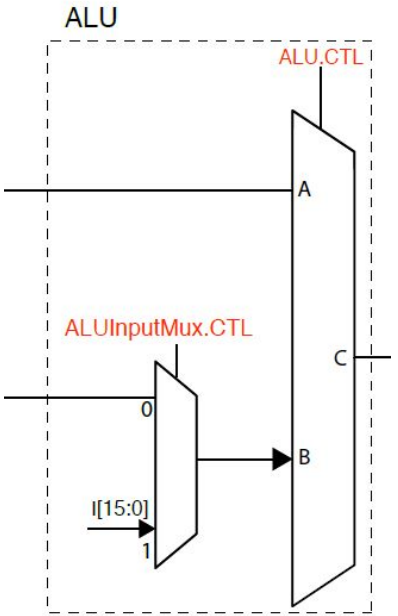
- Stores Registers R0, R1, R2 … R7
  - Registers are “temporary variables”
  - Sometimes have conventions
    - R7 often used to store the program counter
  - Control signals used to specify where to read register addresses from the 16 bit instruction code
- regFile.WE should be enabled iff one of R0 – R7 is being written to, disabled otherwise



Note: “iff” = “if and only if”

Datapath Components: ALU

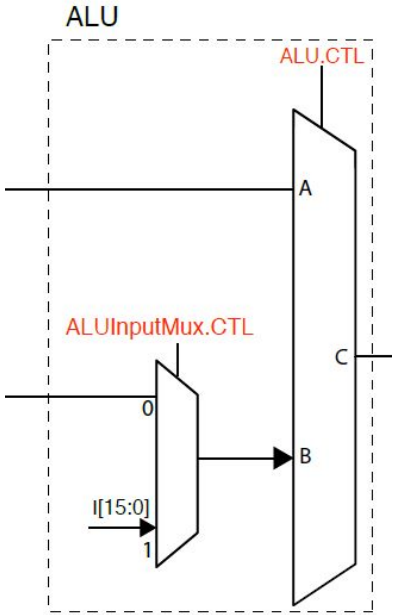
- Arithmetic Logic Unit (ALU)
  - Performs arithmetic operations and logical operations
  - (ALU.CTL) decides which operation is performed
  - ALUInputMux.CTL controls one of the inputs



Datapath Components: ALU

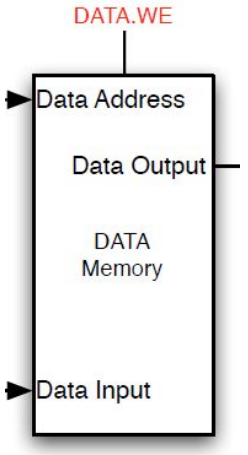
ALUInputMux.CTL	1	0	B[15:0] = RT[15:0] – B input to ALU = RT
		1	B[15:0] = I[15:0] – B input to ALU = Instruction Word

Signal Name	# of bits	Value	Action
ALU.CTL	6		
Arithmetic Ops		0	C = A + B : Addition
		1	C = A * B : Multiplication
		2	C = A - B : Subtraction
		3	C = A / B : Division
		4	C = A % B : Modulus
		5	C = A + SEXT(B[4:0])
		6	C = A + SEXT(B[5:0])
Logical Ops		8	C = A AND B : Bitwise Logical Product
		9	C = NOT A: Bitwise Negation
		10	C = A OR B: Bitwise Logical Sum
		11	C = A XOR B: Bitwise Exclusive OR
		12	C = A AND SEXT(B[4:0])
Comparator Ops		16	C = signed-CC(A-B) [-1, 0, +1]
		17	C = unsigned-CC(A-B) [-1, 0, +1]
		18	C = signed-CC(A-SEXT(B[6:0])) [-1, 0, +1]
		19	C = unsigned-CC(A-SEXT(B[6:0])) [-1, 0, +1]
Shifter Ops		24	C = A << B[3:0] : Shift Left Logical
		25	C = A >>> B[3:0] : Shift Right Arithmetic
		26	C = A >> B[3:0] : Shift Right Logical
Constant Ops		32	C = SEXT(B[8:0])
		33	C = (A & OxFF) | (B[7:0] << 8)



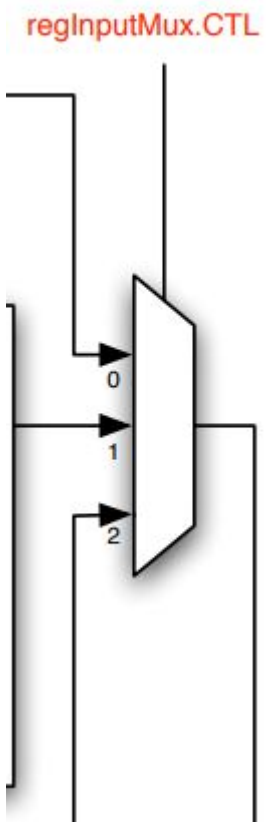
Datapath Components: Data Memory

- Longer term storage than registers
- More difficult to access, need individual instructions to read/write (LDR/STR)
- One Control Signal DATA.WE
  - Should be 1 iff you are **storing** something in memory, 0 otherwise



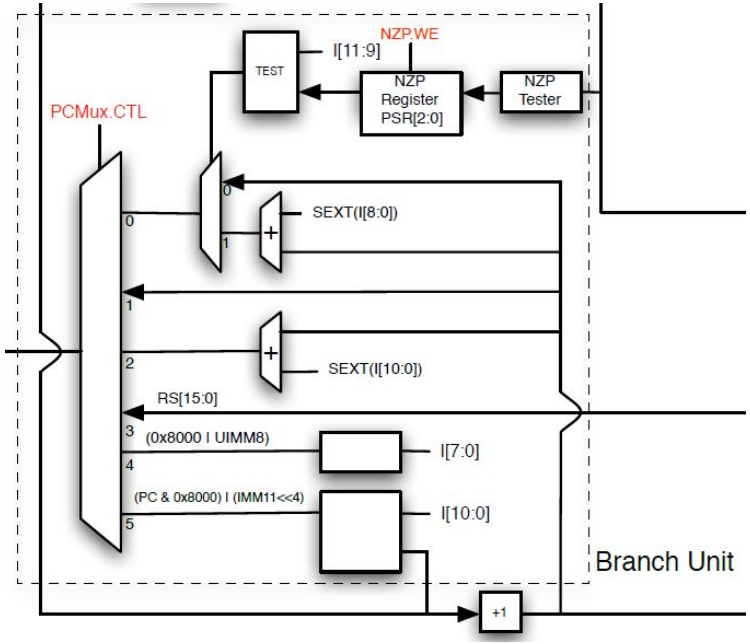
Datapath Components: regInputMux.ctl

- Inputs:
  - 0 -> ALU result (in case we need to store operation result, e.g. ADD, AND, MOD, etc.)
  - 1 -> Data Output (in case we load from memory)
  - 2 -> PC + 1 (in case we need to store PC + 1 into R7)
- Output:
  - Goes into write input of Register File (since we are storing into register file!)



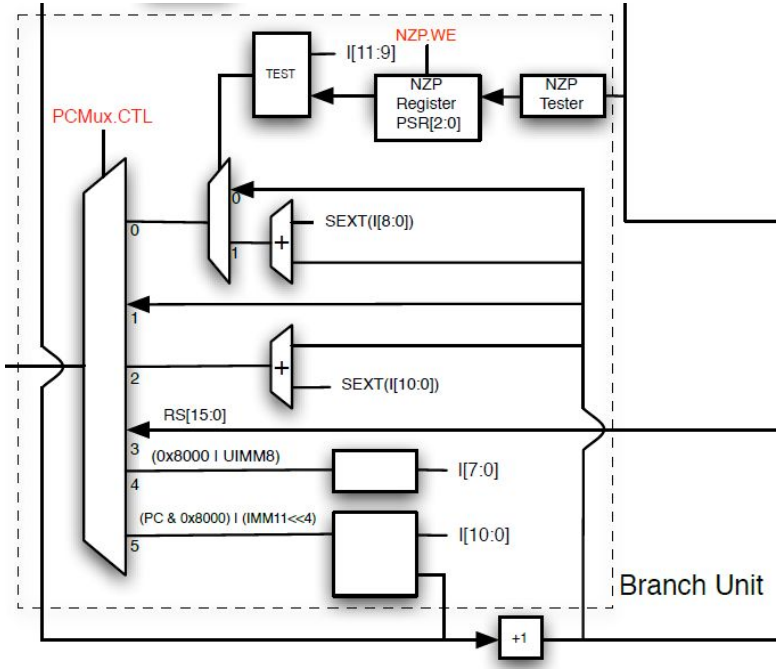
Datapath Components: Branch Unit

- PCMux.CTL
  - "How are we changing PC?"
  - How PC is updated
  - Usually is 1
    (means pc = pc +1)
- NZP.WE
  - "Are we modifying NZP bits?"
  - should be 1 iff regFile.WE is 1 or instruction explicitly sets NZP



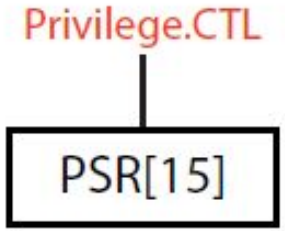
Datapath Components: Branch Unit Breakdown

- PCMux.CTL
  - 0 -> PC = PC + 1 (if not branching)
  - 0 -> PC = PC + 1 + sext(IMM9) (if branching)
  - 1 -> PC = PC + 1
  - 2 -> PC = PC + 1 + sext(IMM11)
  - 3 -> PC = RS
  - 4 -> PC = (0x8000 | UIMM8)
  - 5 -> PC = (PC & 0x8000) | (IMM11 << 4)



Datapath Components: Privilege

- Privilege determines whether we are in “User” mode or “Supervisor” mode.
  - Some things (like Operating System stuff) requires Supervisor mode (More in next week’s lectures)
- Privilege.CTL
  - Decides how the Privilege Bit (which is stored in PSR) should change.
  - Should be 2 most of the time
    - We aren’t changing privilege for most instructions
  - TRAP and RTI
    - Instructions that change PSR[15]



Privilege.CTL	2	0	PSR[15] = 0 – Clear privilege bit
		1	PSR[15] = 1 – Set privilege bit
		2	PSR[15] unchanged – no change to privilege bit

Exercises

- Write out all control signals of the following instructions
  - SUB
  - LDR
  - TRAP
  - RTI
  - CONST
  - BRzp

Control Signals: SUB

SUB Rd Rs Rt	Rd = Rs - Rt
0001 ddds ss01 0ttt	

rsMux.CTL → 0
rtMux.CTL → 0
rdMux.CTL → 0
regFile.WE → 1
ALUInputMux.CTL → 0
ALU.CTL → 2
DATA.WE → 0
regInputMux.CTL → 0
NZP.WE → 1
Privilege.CTL → 2
PCMux.CTL → 1

LDR Rd Rs IMM6	Rd = dmem[Rs + sext(IMM6)]

0110 ddds ssii iiii

Control Signals: LDR

rsMux.CTL → 0
rtMux.CTL → X (not used)
rdMux.CTL → 0
regFile.WE → 0
ALUInputMux.CTL → 1
ALU.CTL → 6
DATA.WE → 0
regInputMux.CTL → 1
NZP.WE → 1
Privilege.CTL → 2
PCMux.CTL → 1

TRAP UIMM8	R7 = PC + 1; PC = (0x8000 | UIMM8); PSR [15] = 1

1111 xxxx uuuu uuuu

Control Signals: TRAP

rsMux.CTL → X
rtMux.CTL → X
rdMux.CTL → 1 (store in R7)
regFile.WE → 1
ALUInputMux.CTL → X
ALU.CTL → X
DATA.WE → 0
regInputMux.CTL → 2 (store PC + 1)
NZP.WE → 1
Privilege.CTL → 1 (PSR[15] = 1)
PCMux.CTL → 4 (0x8000 | UIMM8)

Control Signals: RTI

```
PC = R7; PSR [15] = 0   1000 xxxx xxxx xxxx
```

rsMux.CTL → 1 (retrieve R7)
rtMux.CTL → X
rdMux.CTL → X
regFile.WE → 0
ALUInputMux.CTL → X
ALU.CTL → X
DATA.WE → 0
regInputMux.CTL → X
NZP.WE → 0
Privilege.CTL → 0 (PSR[15] = 0)
PCMux.CTL → 3 (PC = R7)

Control Signals: CONST

CONST Rd IMM9	Rd = sext(IMM9)
1001 dddi iiii iiii	

rsMux.CTL → X
rtMux.CTL → X
rdMux.CTL → 0
regFile.WE → 1
ALUInputMux.CTL → 1
ALU.CTL → 33
DATA.WE → 0
regInputMux.CTL → 0
NZP.WE → 0
Privilege.CTL → 2
PCMux.CTL → 1

```
(  Z|P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)
                        0000 011i iiii iiii
```

Control Signals: BRzp

rsMux.CTL → X
rtMux.CTL → X
rdMux.CTL → X
regFile.WE → 0
ALUInputMux.CTL → X
ALU.CTL → X
DATA.WE → 0
regInputMux.CTL → X
NZP.WE → 0 (not changing NZP, only using them)
Privilege.CTL → 2
PCMux.CTL → 0 (branch or not branch based on NZP)

That's all we have for today!

Reminders:

- TA-lead recitations will take place on
  - Tuesdays 6:30-8:00pm in Moore 100A
  - Wednesday 12:00-1:30pm in Moore 100C
- Check the course website for OH times