Final Review

Introduction to Computer Systems, Fall 2022

Instructor: Travis McGaha

TAs:

Ali Krema
Andrew Rigas
Anisha Bhatia
Audrey Yang
Craig Lee
Daniel Duan
David LuoZhang
Eddy Yang
Ernest Ng
Heyi Liu
Janavi Chadha
Jason Hom
Katherine Wang
Kyrie Dowling
Mohamed Abaker
Noam Elul
Patricia Agnes
Patrick Kehinde Jr.
Ria Sharma
Sarah Luthra
Sofia Mouchtaris

When poll is active, respond at **pollev.com/tqm**

Text **TQM** to **37607** once to join

Which Question would you like to go over next? (This is not an exhaustive list of testable topics)

- Binary in LC4 & C
- CMOS
- Gates
- Flip Flops & Timing
- LC4 Programming
- Control Signals
- C Programming
- C to ASM
- Code Analysis
- Other/General Q&A

Logistics

- Final Exam: **This Thursday 6-8 pm**
  - Details released on the course website
- TA-Led Final Review
  - Tuesday 1 – 3 pm @ Levine 101
  - Different Questions (probably still useful)
- Many Instructor OH posted to the course calendar

Binary in C & LC4

- To get around only having 16-bit integers in LC4, we create a struct that to mimic a 32-bit integer. Can we do 32 bit add by doing addition on each struct field like shown below? Explain your answer. If it doesn't work, briefly explain how to fix it.

```c
typedef struct uint32_st {
  unsigned int lower_bits; // lower 16 bits of number
  unsigned int upper_bits; // upper 16 bits of number
} uint32;

// This function adds the two 32 bit unsigned numbers
// represented by a and b and puts the result in out.
void add32(uint32 *out, uint32 *a, uint32 *b) {
  out->lower_bits = a->lower_bits + b->lower_bits;
  out->upper_bits = a->upper_bits + b->upper_bits;
}
```

Binary in C & LC4

- In LC4, can we have 32 registers instead of 8 while keeping instructions 16 bits? Why or why not?

CMOS, GATES

- Create a circuit that takes in a 3 1-bit inputs A, B and C and outputs a 1 if and only if A and B have the same value and C has a different value from A and B.
  - List the outputs that result in a 1 for the output
  - Create a corresponding CMOS circuit
    - Can assume you have the inverses of the Input bits
  - Create a corresponding gate level circuit

Gate Level Circuit

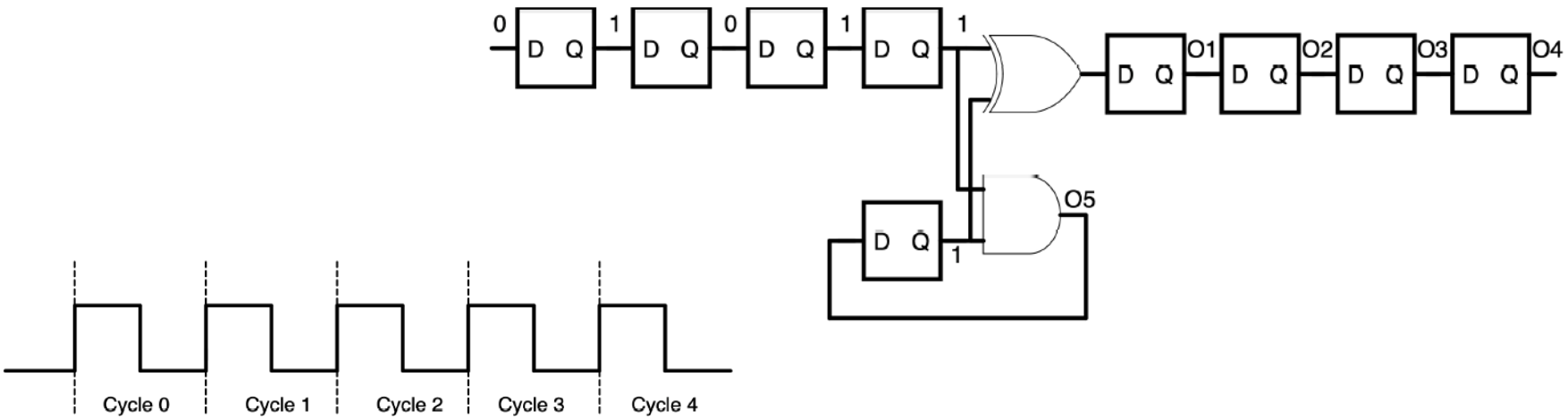
- Create a gate level circuit that takes in 3 1-bit inputs A, B and C and outputs a 1 if and only if A and B have the same value and C has a different value from A and B.

Gates Pt. 2

- Can you make a 2-input XOR using only 3 2-input NANDs and 2 inverters?

Flip Flops & Timing

- The figure shows the values of some of the signals during clock cycle 0, i.e. in the middle of clock cycle 0.
- What are the value of the signals O1 through O5 during the middle of clock cycle 4?
- Note that the input to the leftmost flip-flop is held constant at 0 throughout.



LC4 Programming

- Write an asm program that you can assume has:
  - R0 = addr of start of array
  - R1 = length of array that R0 refers to
  - R2 = addr of start of destination array (same length as R0's array)

- Your asm should calculate the prefix sum and store it in the destination array. Each index in the dest should contain the sum of values up to and including that index in R1. E.g. dest[i] = src[0] + src[1] .. src[i]

Control Signals

- What are the control signals for the new instruction MERGE which takes two registers that you can assume each contain an 8-bit binary value (upper 8-bits are zero-d out) and concatenates one onto the other to make a 16-bit value.
  - Mnemonic: MERGE Rd, Rs, Rt
  - Semantics: Rd = (Rt << 8) + Rs
  - Encoding: 0011 ddds ss00 0ttt
- What are the control signals for this instruction?

C Programming

- Complete the following C program:

```
// Given a string, return the specified substring
// dynamically allocated. The substring should start at
// the specified index of the input string and it's
// length should be the specified length (or until the
// end of the specified string, whatever comes first).
// You may assume 'start' is a valid index, that
// 'str' contains a valid address and 'len' is > 0.
//
// Arguments:
// - str: the str to return the substring of
// - start: the index the substring should start at
// - len: the length of the substring
//
// returns:
// - the string on success, NULL on error
char* substr(char *str, int start, int len) {
  // ...
}
```

C to ASM

- The following code was put through the LCC Compiler

```
void memset(int *ptr, int n, int v) {
  int i;
  for (i = 0; i < n; ++i, ++ptr){
    *ptr = v;
  }
}
```

- The resulting assembly is posted on the course website for this lecture called “memset.asm”, several of the assembly instructions are marked as “MISSING INSN”. For each missing instruction, identify what it should be.

Code Analysis

```
typedef struct {
  int elt[8];
} vec8;
```

- The two functions shown to the right do the same job.
  - Which is shorter in terms of C code?
  - Which is shorter in terms of LC4 instructions?

```
void func1(vec8 *out, vec8 *a, vec8 *b) {
  int i;
  for (i=0; i < 8; ++i)
    out->elt[i] = a->elt[i] + b->elt[i];
}
```

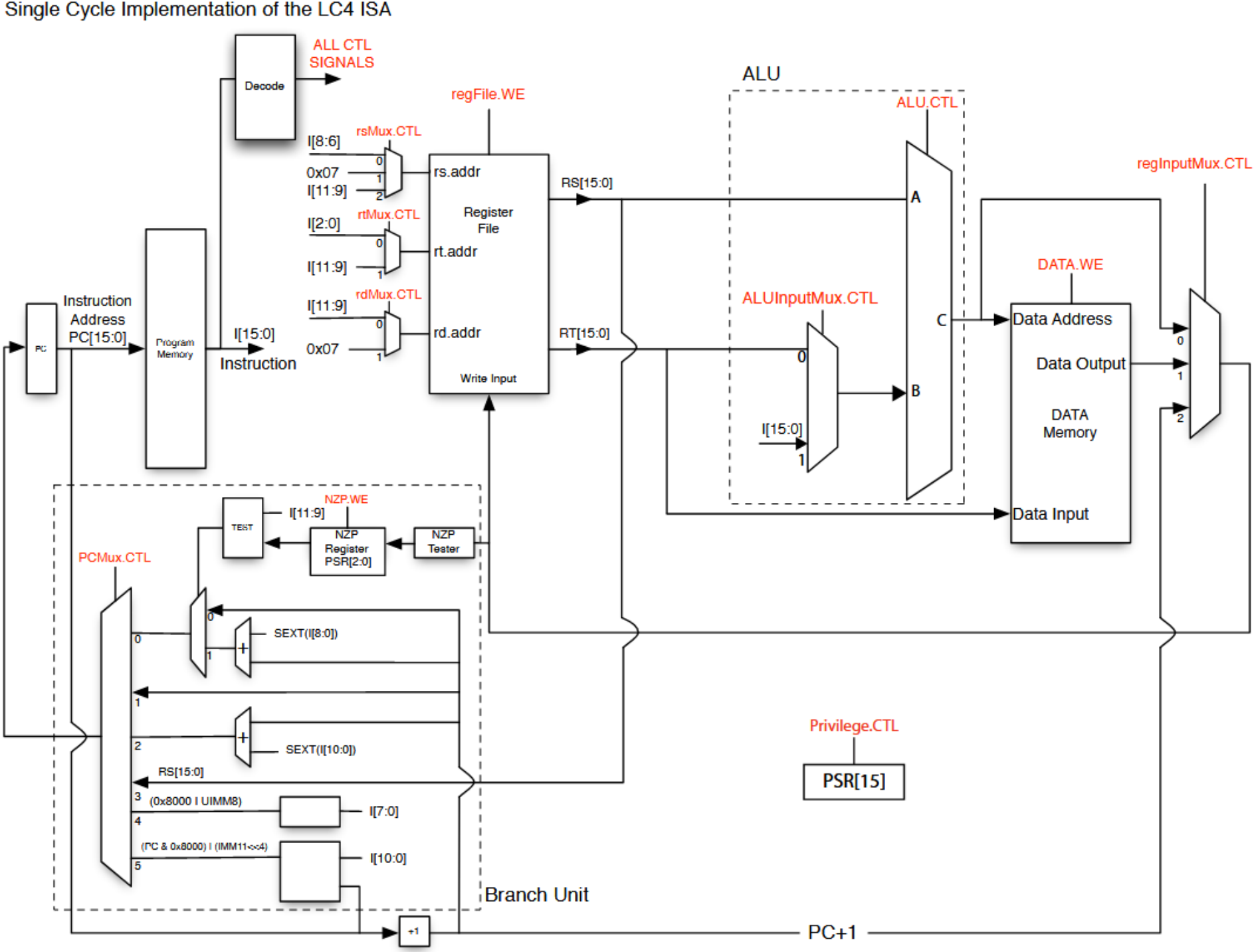
```
void func2 (vec8 *out, vec8 *a, vec8 *b) {
  out->elt[0] = a->elt[0] + b->elt[0];
  out->elt[1] = a->elt[1] + b->elt[1];
  out->elt[2] = a->elt[2] + b->elt[2];
  out->elt[3] = a->elt[3] + b->elt[3];
  out->elt[4] = a->elt[4] + b->elt[4];
  out->elt[5] = a->elt[5] + b->elt[5];
  out->elt[6] = a->elt[6] + b->elt[6];
  out->elt[7] = a->elt[7] + b->elt[7];
}
```

  - If you were to compile each function with lcc and then run the two functions which one do you think would run faster? That is, which one would complete the task in the shortest amount of time. Please explain why

General Questions & Answers

- Take questions/requests from students

Mnemonic	Semantics	Encoding
NOP	PC = PC + 1	0000 000x xxxx xxxx
BRp <Label>	(P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 001i iiii iiii
BRz <Label>	(Z) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 010i iiii iiii
BRzp <Label>	(Z|P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 011i iiii iiii
BRn <Label>	(N) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 100i iiii iiii
BRnp <Label>	(N | P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 101i iiii iiii
BRnz <Label>	(N|Z) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 110i iiii iiii
BRnzp <Label>	(N|Z|P) ? PC = PC + 1 + (sext(IMM9) offset to <Label>)	0000 111i iiii iiii
ADD Rd Rs Rt	Rd = Rs + Rt	0001 ddds ss00 0ttt
MUL Rd Rs Rt	Rd = Rs * Rt	0001 ddds ss00 1ttt
SUB Rd Rs Rt	Rd = Rs - Rt	0001 ddds ss01 0ttt
DIV Rd Rs Rt	Rd = Rs / Rt	0001 ddds ss01 1ttt
ADD Rd Rs IMM5	Rd = Rs + sext(IMM5)	0001 ddds ss1i iiii
MOD Rd Rs Rt	Rd = Rs % Rt	1010 ddds ss11 xttt
AND Rd Rs Rt	Rd = Rs & Rt	0101 ddds ss00 0ttt
NOT Rd Rs	Rd = ~Rs	0101 ddds ss00 1xxx
OR Rd Rs Rt	Rd = Rs | Rt	0101 ddds ss01 0ttt
XOR Rd Rs Rt	Rd = Rs ∧ Rt	0101 ddds ss01 1ttt
AND Rd Rs IMM5	Rd = Rs & sext(IMM5)	0101 ddds ss1i iiii
LDR Rd Rs IMM6	Rd = dmem[Rs + sext(IMM6)]	0110 ddds ssii iiii
STR Rt Rs IMM6	dmem[Rs + sext(IMM6)] = Rt	0111 ttts ssii iiii
CONST Rd IMM9	Rd = sext(IMM9)	1001 dddi iiii iiii
HICONST Rd UIMM8	Rd = (Rd & 0xFF) | (UIMM8 << 8) [1]	1101 ddd x uuuu uuuu
CMP Rs Rt	NZP = sign(Rs - Rt) [2]	0010 sss0 0xxx xttt
CMPU Rs Rt	NZP = sign(uRs - uRt) [3]	0010 sss0 1xxx xttt
CMPI Rs IMM7	NZP = sign(Rs - IMM7)	0010 sss1 0iii iiii
CMPIU Rs UIMM7	NZP = sign(uRs - UIMM7)	0010 sss1 1uuu uuuu
SLL Rd Rs UIMM4	Rd = Rs << UIMM4	1010 ddds ss00 uuuu
SRA Rd Rs UIMM4	Rd = Rs >>> UIMM4	1010 ddds ss01 uuuu
SRL Rd Rs UIMM4	Rd = Rs >> UIMM4	1010 ddds ss10 uuuu
JSRR Rs	R7 = PC + 1; PC = Rs	0100 0xxs ssxx xxxx
JSR <Label>	R7 = PC + 1; PC = (PC & 0x8000) | ((IMM11 offset to <Label>) << 4)	0100 1iii iiii iiii
JMPR Rs	PC = Rs	1100 0xxs ssxx xxxx
JMP <Label>	PC = PC + 1 + (sext(IMM11) offset to <Label>)	1100 1iii iiii iiii
TRAP UIMM8	R7 = PC + 1; PC = (0x8000 | UIMM8); PSR [15] = 1	1111 xxxx uuuu uuuu
RTI	PC = R7; PSR [15] = 0	1000 xxxx xxxx xxxx


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]
rtMux.CTL
I[2:0]
I[11:9]
rdMux.CTL
I[11:9]
0x07
rs.addr
Register File
rt.addr
rd.addr
Write Input
RS[15:0]
RT[15:0]
regInputMux.CTL
A
B
C
ALUInputMux.CTL
I[15:0]
DATA.WE
Data Address
Data Output
DATA Memory
Data Input
PC
Instruction Address PC[15:0]
Program Memory
I[15:0]
Instruction
NZP.WE
I[11:9]
TEST
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]
Branch Unit
+1
PC+1
Privilege.CTL
PSR[15]

Signal Name	# of bits	Value	Action
PCMux.CTL	3	0	Value of NZP register compared to bits I[11:9] of the current instruction if the test is satisfied then the output of TEST is 1 and NextPC = BRANCH Target, (PC+1) + SEXT(IMM9); otherwise the output of TEST is 0 and NextPC = PC + 1
		1	Next PC = PC+1
		2	Next PC = (PC+1) + SEXT(IMM11)
		3	Next PC = RS
		4	Next PC = (0x8000 | UIMM8)
		5	Next PC = (PC & 0x8000) | (IMM11 << 4)
rsMux.CTL	2	0	rs.addr = I[8:6]
		1	rs.addr = 0x07
		2	rs.addr = I[11:9]
rtMux.CTL	1	0	rt.addr = I[2:0]
		1	rt.addr = I[11:9]
rdMux.CTL	1	0	rd.addr = I[11:9]
		1	rd.addr = 0x07
regFile.WE	1	0	Register file not written
		1	Register file written: rd.addr indicates which register is updated with the value on the Write Input
regInput.Mux.CTL	2	0	Write Input = ALU output
		1	Write Input = Output of Data Memory
		2	Write Input = PC + 1
NZP.WE	1	0	NZP register not updated
		1	NZP register updated from Write Input to register file
DATA.WE	1	0	Data Memory not written
		1	Data Input written into location on Data Address lines
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

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)