CS61C Summer 2013
Final Review Session

Albert, Jeffrey, Justin, Kevin, Sagar, Shaun

Please sit as close to the front as possible;
some slides contain small text

Reminders

- Time/Location: Friday, August 16th, 9am-12pm in 155 Dwinelle
- Rules:
  - **Bring**: Pencil, Eraser, TWO double-sided sheets of handwritten notes
  - **Provided**: MIPS Green Sheet, exam with whitespace for work, scratch paper, time keeping
  - ***DO NOT* Bring**: Books, printed notes, calculators

- Additional resources available on Piazza @893

Outline

1. VM and Cache
2. SIMD and OpenMP
3. FSMs, Boolean Logic, and Timing
4. Datapath and Control
5. Pipelining and Hazards
6. MapReduce

VM and Cache (1/5)

Machine Specs (one level of Cache and VM):

1 MiB Physical Addr. Space

4 GiB Virtual Addr. Space

4 KiB Page Size

16 KiB 8-way set assoc., write-through cache w/LRU

1 KiB cache block size

2-entry TLB, LRU

a) What is the T:I:O bit breakup for the cache (assuming byte addressing)?

b) What is the VPN: PO bit breakup for VM (assuming byte addressing)?

VM and Cache (2/5)

Machine Specs (one level of Cache and VM):

1 MiB Physical Addr. Space, 4 GiB Virtual Addr. Space, 4 KiB Page Size, 16 KiB 8-way set assoc., write-through cache w/LRU, 1 KiB cache block size, 2-entry TLB, LRU

The following code is run on the system:

```
#define NUM_INTS 8192
int *A = (int *)malloc(NUM_INTS * sizeof (int));
int i, total = 0;
for(i = 0; i < NUM_INTS; i+=128) A[i] = i;
for(i = 0; i < NUM_INTS; i+=128) total += A [i]; // SPECIAL
```

VM and Cache (3/5)

Machine Specs (one level of Cache and VM):

1 MiB Physical Addr. Space, 4 GiB Virtual Addr. Space, 4 KiB Page Size, 16 KiB 8-way set assoc., write-through cache w/LRU, 1 KiB cache block size, 2-entry TLB, LRU

The following code is run on the system:

```
#define NUM_INTS 8192
int *A = (int *)malloc(NUM_INTS * sizeof(int));
int i, total = 0;
for(i = 0; i < NUM_INTS; i+=128) A[i] = i;
for(i=0; i<NUM_INTS; i+=128) total+=A[i]; //ME
```

c) Calculate the hit percentage for the cache

VM and Cache (4/5)

Machine Specs (one level of Cache and VM):

1 MiB Physical Addr. Space, 4 GiB Virtual Addr. Space, 4 KiB Page Size, 16 KiB 8-way set assoc., write-through cache w/LRU, 1 KiB cache block size, 2-entry TLB, LRU

The following code is run on the system:

```
#define NUM_INTS 8192
int *A = (int *)malloc(NUM_INTS * sizeof(int));
int i, total = 0;
for(i = 0; i < NUM_INTS; i+=128) A[i] = i;
for(i=0; i<NUM_INTS; i+=128) total+=A[i]; //ME
```

d) Calculate the hit percentage for the TLB

VM and Cache (5/5)

Machine Specs (one level of Cache and VM):

1 MiB Physical Addr. Space, 4 GiB Virtual Addr. Space, 4 KiB Page Size, 16 KiB 8-way set assoc., write-through cache w/LRU, 1 KiB cache block size, 2-entry TLB, LRU

The following code is run on the system:

```
#define NUM_INTS 8192
int *A = (int *)malloc(NUM_INTS * sizeof(int));
int i, total = 0;
for(i = 0; i < NUM_INTS; i+=128) A[i] = i;
for(i=0; i<NUM_INTS; i+=128) total+=A[i]; //ME
```

e) Calculate the hit percentage for the page table

SIMD (1/6)

Remember Polynomial Differentiation:

$$\frac{d}{dx}(a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}) = a_1 + 2a_2x + \cdots + (n-1)a_{n-1}x^{n-2}$$

A degree n - 1 polynomial is uniquely determined by n integer coefficients a_0 … a_(n-1), which we can represent as an array of integer values. For example, f(x) = 1 + 3x^2 can be represented by:

int f[] = {1, 0, 3};

Taking the derivative of a polynomial reduces its degree by 1 (Hint: What are the implications for the length of the array into which we store the derivative?)

SIMD (2/6)

Suppose we have the following SIMD-ized code to differentiate the given polynomial, four terms at a time:

```
void differentiate_SIMD(int32_t *A_prime, int32_t *A, size_t n) {
    int i, j = J_INIT;
    int32_t tmp[] = STEROIDS_INIT;
    __m128i j_on_steroids = _mm_loadu_si128(tmp);

    differentiate(A_prime, A, START + 1) // given non-SIMD to handle fringe

    j += START; // translate j, j_on_steroids by START
    j_on_steroids = _mm_add_epi32(j_on_steroids, _mm_set1_epi32(START));

    for (i = START; i < END; i += 4) {  // main loop
        __m128i A_chunk = _mm_loadu_si128(&A[j]);
        _mm_storeu_si128(&A_prime[i],_mm_mul_epi32(j_on_steroids,A_chunk));
        j += 4;
        j_on_steroids = _mm_add_epi32(j_on_steroids,!_mm_set1_epi32(4));
    }
}
```

SIMD (3/6)

i) For the given code, select the correct definition for the macro START:

a) #define START 4
b) #define START ((n-1)/4*4)
c) #define START 0
d) #define START (n % 4)
e) #define START ((n-1) % 4)

SIMD (4/6)

ii) For the given code, select the correct definition for the macro J_INIT:

a) #define J_INIT 0
b) #define J_INIT 1
c) #define J_INIT 2
d) #define J_INIT 4
e) #define J_INIT 5

SIMD (5/6)

iii) For the given code, select the correct definition for the macro STEROIDS_INIT:

a) #define STEROIDS_INIT {0, 0, 0, 0}
b) #define STEROIDS_INIT {1, 1, 1, 1}
c) #define STEROIDS_INIT {0, 1, 2, 3}
d) #define STEROIDS_INIT {1, 2, 3, 4}
e) #define STEROIDS_INIT {4, 4, 4, 4}

SIMD (6/6)

iv) For the given code, select the correct definition for the macro END:

a) #define END (n - 1)
b) #define END (n)
c) #define END ((n-1)/4*4)
d) #define END (n/4*4)
e) #define END (n/4)

OpenMP (1/2)

Circle the one choice that results in the fastest parallel code with the same output as the initial serial code. For convenience: ogtn is the same as omp_get_thread num and ognt is the same as omp_get_num_threads

Note: parallel/critical block around "..." => #pragma omp parallel/critical { … }

i) Serial:

```
int i;
for (i = 0; i < len_list; i += 1)
   total += list[i];
for (i = 0; i < len_result; i += 1)
   result[i] = total*i;
```

Parallel:

```
1 for (int i = ogtn(); i < len_list; i += ognt())
2   total += list[i];
3 for (int i = ogtn(); i < len_result; i += ognt())
4   result[i] = total*i;
```

a) Add a parallel block around lines 1-2 and a parallel block around lines 3-4

b) Same as in (a) and additionally, add critical block around line 2

c) Add a parallel block around lines 1-4 and a critical block around line 2

d) Add a parallel block around lines 1-4

e) None of the above cause the code to return the correct result.

OpenMP (2/2)

Circle the one choice that results in the fastest parallel code with the same output as the initial serial code. For convenience: ogtn is the same as omp_get_thread num and ognt is the same as omp_get_num_threads

Note: parallel/critical block around "..." => #pragma omp parallel/critical { … }

ii) Serial:

```
for(int i = 0; i < m; i += 1) {
    total += i;
    res[i] = total;
}
```

Parallel:

```
0 #pragma omp parallel
1 {
2   for(int i = ogtn(); i < m; i += ognt())
3   {
4       total += i;
5       res[i] = total;
6   }
7 }
```

a) Add a critical block around lines 2-6

b) Add a critical block around lines 4-5

c) Add a critical block around line 4

d) The parallel code returns the correct result the fastest without adding anything

e) None of the above cause the code to return the correct result.

FSMs, Boolean Logic, Timing (1/6)

You are an intern at a massive hardware firm. Your first task is to design an “odd counter” circuit that receives a single bit input every cycle and outputs a single bit every cycle. It outputs a 1 if and only if it has seen an odd number of ones AND an odd number of zeros. It starts in a state where it has seen an even number of ones and an even number of zeros (remember, zero is an even number). As an example,

the input:

```
I: 1 1 0 1 1 1 0 0 0 1 0 1 1 0 0
```

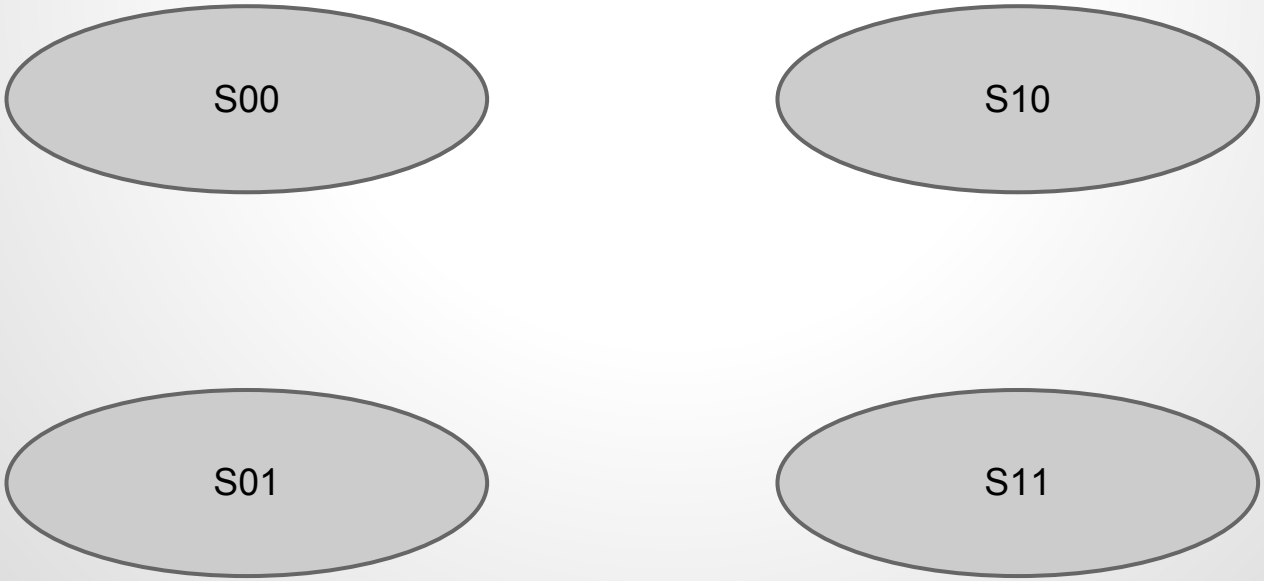
will produce the output:

```
O: 0 0 0 1 0 1 0 1 0 0 0 1 0 0 0
```

FSMs, Boolean Logic, Timing (2/6)

The FSM outputs a 1 if and only if it has seen an odd number of ones AND an odd number of zeroes.

a) Complete the FSM diagram:



FSMs, Boolean Logic, Timing (3/6)

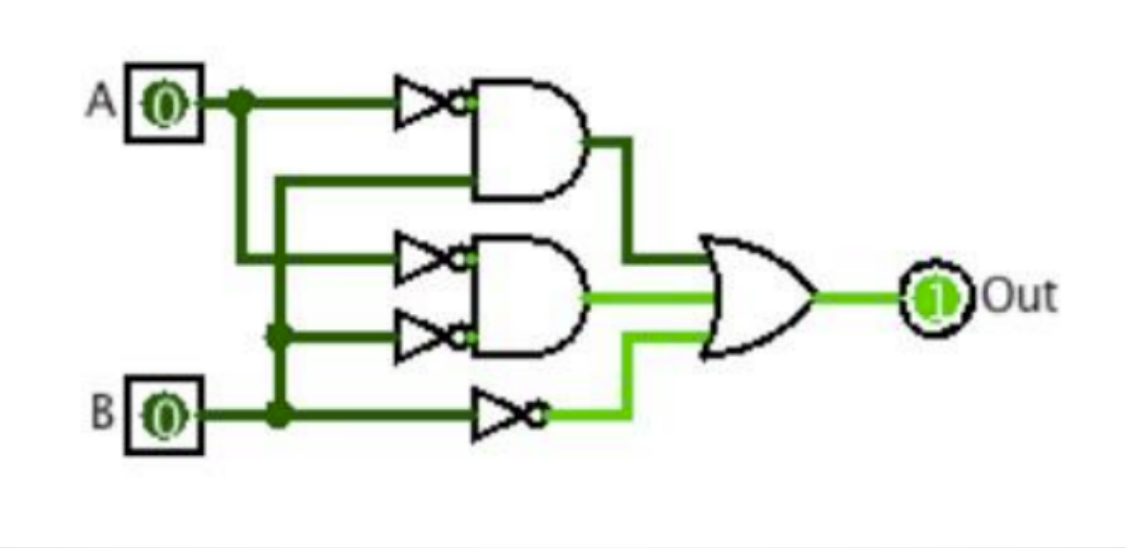
The FSM outputs a 1 if and only if it has seen an odd number of ones AND an odd number of zeroes.

b) Complete the truth table:

P1	P0	I	O	N1	N0
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

FSMs, Boolean Logic, Timing (4/6)

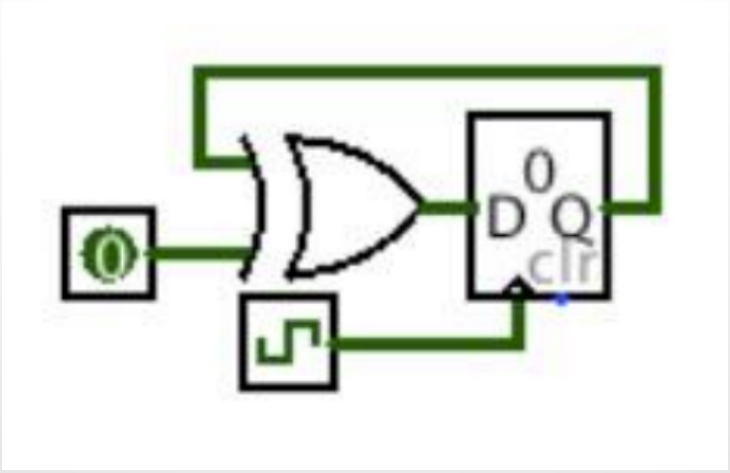
c) Rebuild this circuit with the fewest gates, using ONLY AND, OR and NOT gates.



FSMs, Boolean Logic, Timing (5/6)

d) Your boss wants you to choose an XOR gate for the circuit below: The clock speed is 2GHz, the setup, hold, and clock-to-q times of the register are 40, 70, and 60 picoseconds (10-12s) respectively.

What range of XOR gate delays is acceptable?
E.g., “at least W ps”, “at most X ps”, or “Y to Z ps”.

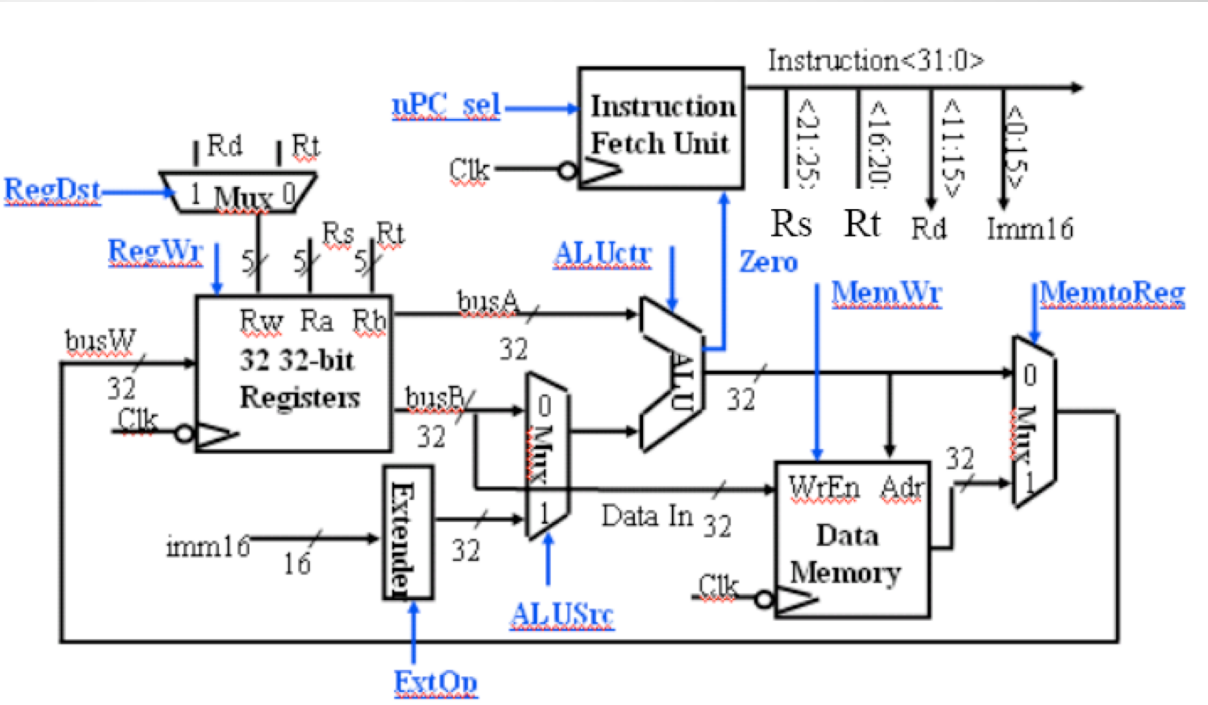


FSMs, Boolean Logic, Timing (6/6)

e) You're asked to create all the unique 3-to-2 circuits (i.e., 3 inputs: I2,I1,I0 and 2 outputs), with one minor catch. Your circuit must ignore the value of I1 if the value of I2 is 1. How many different circuits will you have to make? Use IEC terminology, like 128 mebicircuits, 512 tebicircuits, etc.

Datapath and Control (1/5)

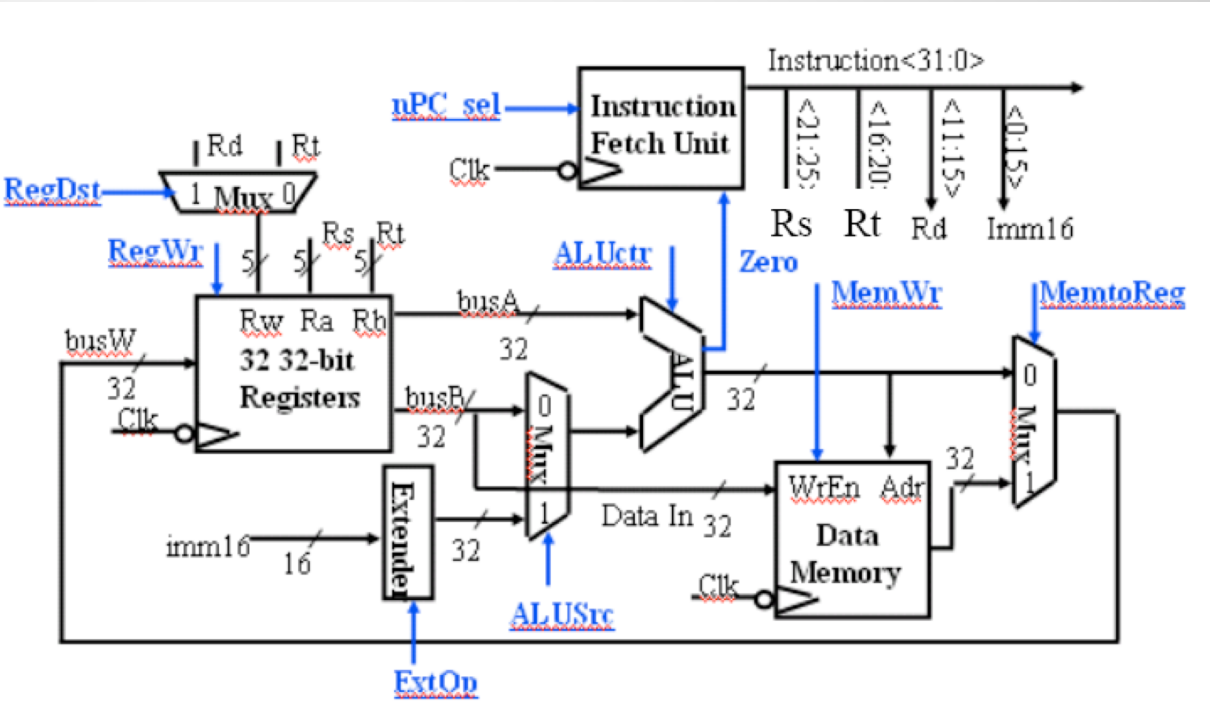
On the right is the single-cycle MIPS datapath (no pipelining). Modify the diagram using adders, shifters, mux chips, wires, and new control signals to implement `addpr`. If necessary you may replace original labels.



`addpr` - almost identical to `addi`, but: stores into the `rt` register, the sum of the immediate and the value of the register specified by the lowest 5 bits in memory at the address specified by the pointer stored in the `rt` register.

Datapath and Control (2/5)

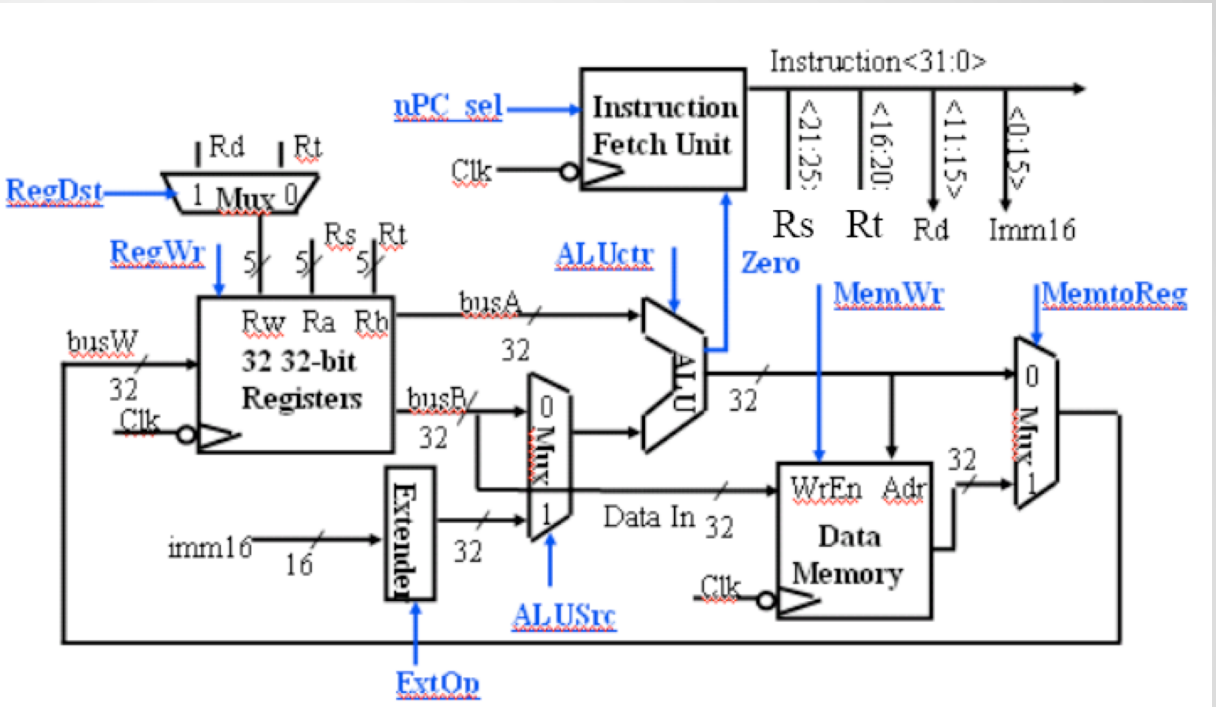
On the right is the single-cycle MIPS datapath (no pipelining). Modify the diagram using adders, shifters, mux chips, wires, and new control signals to implement `addpr`. If necessary you may replace original labels.



a) Make up the syntax for the I-type MAL MIPS instruction that does it (show an example if the pointer lives in $v0, and the constant is 5). On the right, show the register transfer language (RTL) description of `addpr`

Datapath and Control (3/5)

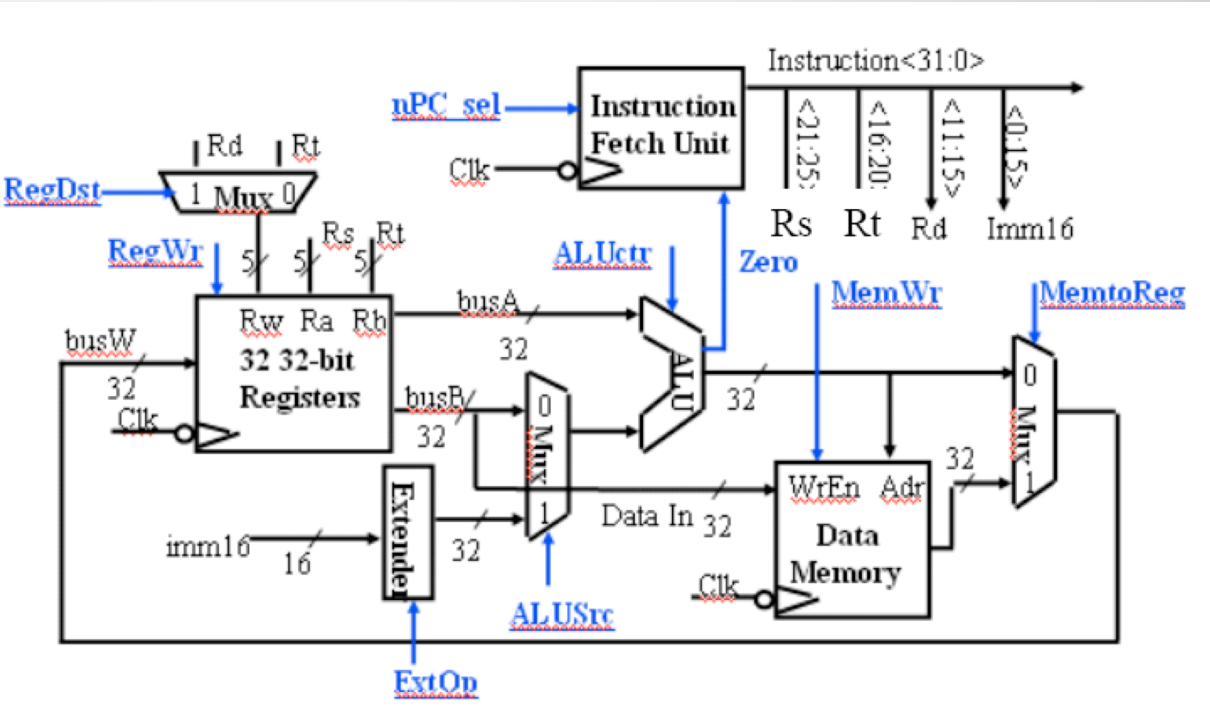
On the right is the single-cycle MIPS datapath (no pipelining). Modify the diagram using adders, shifters, mux chips, wires, and new control signals to implement `addpr`. If necessary you may replace original labels.



b) Change as little as possible in the datapath above and list all changes.

Datapath and Control (4/5)

On the right is the single-cycle MIPS datapath (no pipelining). Modify the diagram using adders, shifters, mux chips, wires, and new control signals to implement `addpr`. If necessary you may replace original labels.

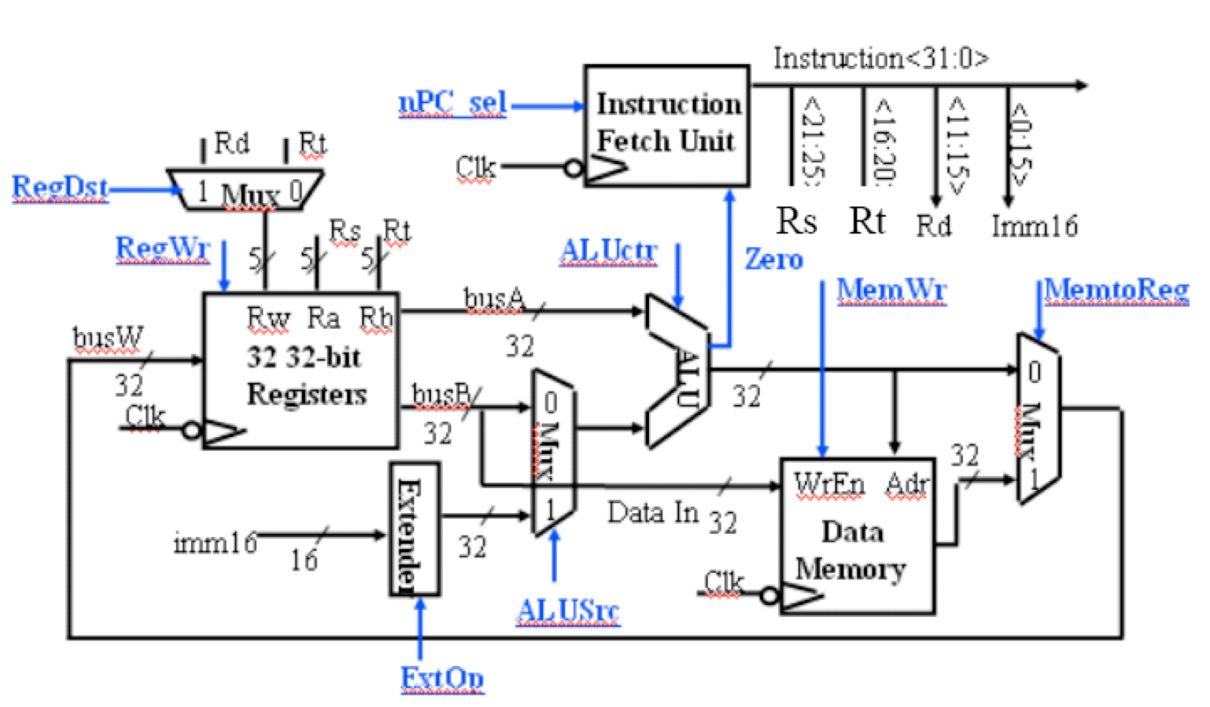


c) We now want to set all the control lines appropriately. List what each signal should be, an intuitive name or {0, 1, x for don't care}. Include any new control signals you added.

`RegDst, RegWr, nPC_sel, ExtOp, ALUSrc, ALUctr, MemWr, MemtoReg`

Datapath and Control (5/5)

d) In the context of a single-cycle CPU, `lw` used to be the critical path instruction, since it utilized the most components of our datapath. Use the terms below to create an expression clock period will be if we also consider the `addpr` instruction:



`PCRegClkToQ, InstMemAccess, ControlLogicDelay, RegFileAccess, ALUDelay, DataMemAccess, RegSetup, RegHold, MemSetup, MemHold`

Pipelining and Hazards (1/5)

Consider the 5-stage single-issue pipelined MIPS datapath consisting of Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Mem, and Write-Back (WB).

You are given:

```
# $s0 to $s3 = 56, 30, 30, 7
# $t0 to $t4 = 7, 7, 7, 7, 7
add $t0, $s0, $0
and $t1, $t0, $s1
or  $t2, $t0, $s2
sub $t3, $t0, $s3
srl $t4, $t0, 2
```

Start numbering the cycles with 1 when the add instruction enters the IF stage.

i) For this part, assume that the datapath is broken and there is no forwarding / stalling. What are the values of $t0 to $t4 at the end of cycle 7?

Pipelining and Hazards (2/5)

Consider the 5-stage single-issue pipelined MIPS datapath consisting of Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Mem, and Write-Back (WB).

You are given:

```
# $s0 to $s3 = 56, 30, 30, 7
# $t0 to $t4 = 7, 7, 7, 7, 7
add $t0, $s0, $0
and $t1, $t0, $s1
or  $t2, $t0, $s2
sub $t3, $t0, $s3
srl $t4, $t0, 2
```

Start numbering the cycles with 1 when the add instruction enters the IF stage.

ii) For this part, assume that the datapath is broken and there is no forwarding / stalling. What are the values of $t0 to $t4 at the end of cycle 8?

Pipelining and Hazards (3/5)

Consider the 5-stage single-issue pipelined MIPS datapath consisting of Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Mem, and Write-Back (WB).

You are given:

```
# $s0 to $s3 = 56, 30, 30, 7
# $t0 to $t4 = 7, 7, 7, 7, 7
add $t0, $s0, $0
and $t1, $t0, $s1
or  $t2, $t0, $s2
sub $t3, $t0, $s3
srl $t4, $t0, 2
```

Start numbering the cycles with 1 when the add instruction enters the IF stage.

iii) For this part, assume that the datapath is broken and there is no forwarding / stalling. What are the values of $t0 to $t4 at the end of cycle 9?

Pipelining and Hazards (4/5)

Consider the 5-stage single-issue pipelined MIPS datapath consisting of Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Mem, and Write-Back (WB).

You are given:

```
# $s0 to $s3 = 56, 30, 30, 7
# $t0 to $t4 = 7, 7, 7, 7, 7
add $t0, $s0, $0
and $t1, $t0, $s1
or  $t2, $t0, $s2
sub $t3, $t0, $s3
srl $t4, $t0, 2
```

Start numbering the cycles with 1 when the add instruction enters the IF stage.

iv) What instruction (s) is/are computing the wrong result(s)?

Pipelining and Hazards (5/5)

Consider the 5-stage single-issue pipelined MIPS datapath consisting of Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Mem, and Write-Back (WB).

You are given:

```
# $s0 to $s3 = 56, 30, 30, 7
# $t0 to $t4 = 7, 7, 7, 7, 7
add $t0, $s0, $0
and $t1, $t0, $s1
or  $t2, $t0, $s2
sub $t3, $t0, $s3
srl $t4, $t0, 2
```

Start numbering the cycles with 1 when the add instruction enters the IF stage.

v) Suppose we want to fix the problem from part (iv) using forwarding. Which forwarding paths do we need to provide in order to execute the code correctly?

MapReduce (1/2)

The moving average (a type of low-pass filter) is an operation commonly used to smooth noisy data. Here we compute a centered moving average of width WIDTH on an array of data of size SIZE, where each element in our output array is the average of the current element, the previous (WIDTH-1)/2 elements, and the next (WIDTH-1)/2 elements. Assume that WIDTH is odd for simplicity and use zeroes where “required” elements do not exist.

Example Input: float[] A = [7, 2, 3, 4, 8, 6]

Output for WIDTH=3: float[] result = [3, 4, 3, 5, 6, 4.6666]

MapReduce (2/2)

Fit this problem to the MapReduce paradigm. You may assume that you have access to the global variables WIDTH and SIZE:

```
// receives data one elem at a time. key is index i, val is A[i]
map (int key, float value) {
    _____________________________________ {
        context.write(x, value);
    }
}

//outputs must be of the form: key is index i, val is moving
avg.
reduce (int key, float[] values) {
    float total = 0;
    // do not emit keys that do not exist in output array
    if ( (key >= 0) && (key < SIZE) ) {
        ____________________________________
        ____________________________________
        context.write ( ____________________, ____________________);
    }
}
```

That's it for this session!

Good luck on your final!

Acknowledgements

1. VM and Cache - Fa06 Final
2. SIMD and OpenMP - Fa12 Final
3. FSMs, Boolean Logic, and Timing - Sp08 Final
4. Datapath and Control - Sp07 Final
5. Pipelining and Hazards - Fa12 Final
6. MapReduce - Sp13 Final