CS 61C Final review session

Topics (survey results - % of students who want the topic to be covered)

- Caches (79% of students)
- CPU Datapath & Control (71% of students)
- CPU Pipelining (93% of students)
- MIPS (36% of students)

TA: Hokeun Kim

Types of Caches

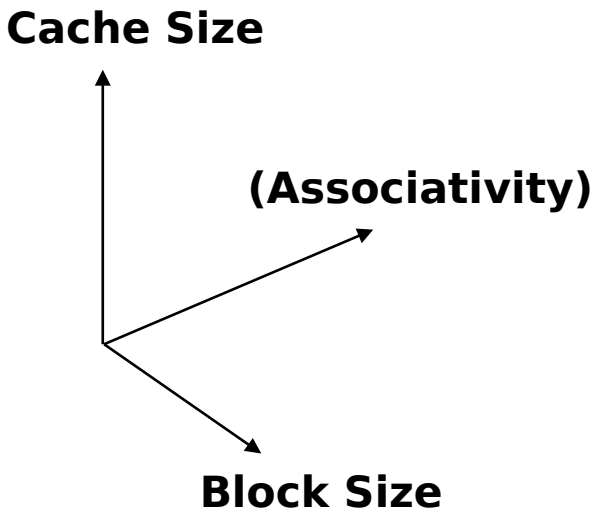
- Direct-mapped caches
  - Each block maps into a single place (row)
- Fully associative caches
  - A block maps into anywhere
- N-way set associative caches
  - Each block maps into N places (rows)
  - Maps into a set, and N rows per set

Sources of Cache Misses: The 3Cs

- Compulsory
  - Misses for 1st reference of data
- Capacity
  - Caused because a cache cannot contain all necessary blocks
- Conflict
  - Multiple memory blocks mapped to the same cache location

Cache Design Space

- Cache parameters
  - Cache size
  - Block size
  - Associativity



Impact of changing cache parameters

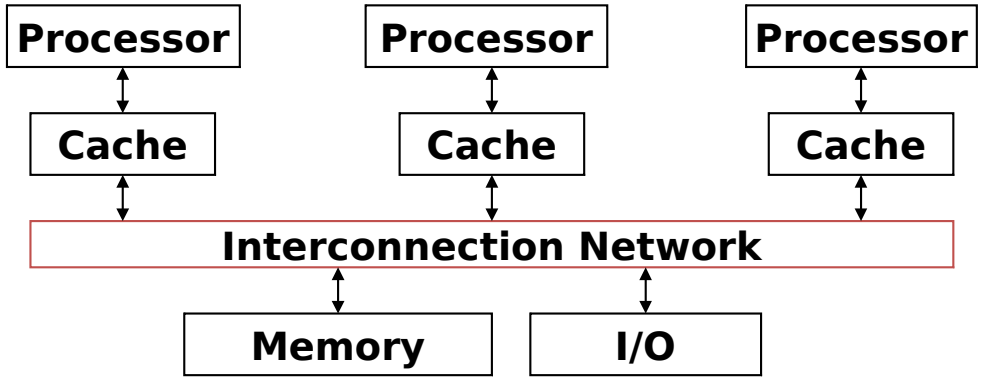
- Larger cache size
  + Less capacity misses (cache can contain more data)
  - Longer hit time (longer signal path)
- Larger block size
  + Less compulsory misses (less misses to get the same amount of cold data)
  - Higher miss penalty (loads more data at each miss)
  - May increase conflict misses
- Higher set associativity
  + Less conflict misses (more blocks per set, less collisions)
  - Longer hit time (more tags to be matched)

Average Memory Access Time (AMAT)

- AMAT = Hit time + Miss rate × Miss penalty
- Reducing hit time
  - Smaller caches (A main reason for why we have small L1 caches)
- Reducing miss rate
  - Find an optimal point with three cache parameters (cache size, block size, and set associativity)
- Reducing miss penalty
  - Add lower level (L2 or L3) caches
  - Getting data from lower level (L2 or L3) caches rather than going all the way to the memory

Cache coherence

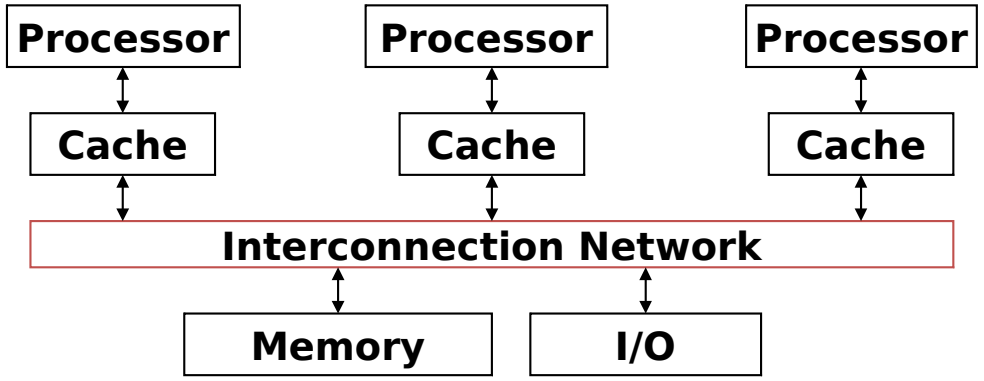
- Shared memory multiprocessor



- Cache coherence
  - When the same memory block is cached by multiple processors, we need to ensure everyone sees the same data
  - What happens when a processor writes on a shared memory block?
  - The memory block on other caches must be invalidated (Cache coherence – write invalidate protocol)
  - When a processor accesses an invalidated block, it gets a cache miss and should read the new data from memory (coherence miss)

Cache coherence

- Shared memory multiprocessor



- False sharing problem
  - Both variables x and y happen to be in the same cache block
  - Proc0 writes on x while Proc1 writes on y
  - What is the problem with this?
  - The cache block has to be invalidated even though they're not writing on the same data
  - False sharing!
  - Unwanted cache misses for the invalidated block (recall lab8 exercise 1)

Cache questions

- Summer '14 Midterm
  - 32-bit byte addressed MIPS
  - A two-way set associative 16KiB cache, a write-back policy, and 64B blocks
  - Assume a list of arrays containing 960 (i.e. 15*64 ints) elements

- The size of struct list is 64B

```
#define ARRAYNESS 15
typedef struct list {
  int vals[ARRAYNESS];
  struct list *next;
} list;
```

```
/* Version 1 */
void copy1(list *dst, list *src) {
  int i;
  while(src) {
    for (i=0; i<ARRAYNESS; i++)
      dst->vals[i] = src->vals[i];
    src = src->next;
    dst = dst->next;
  }
}
```

- Version1: Copy the whole array first, then move on to the next node in the linked list

```
/* Version 2 */
void copy2(list *dst, list *src) {
  int i;
  for (i=0; i<ARRAYNESS; i++) {
    lst *src2 = src, *dst2 = dst;
    while(src2) {
      dst2->vals[i] = src2->vals[i];
      src2 = src2->next;
      dst2 = dst2->next;
    }
  }
}
```

- Version2: Copy the i_{th} elements of each array through the linked list, then move on the $i+1_{th}$ elements of each array

Cache questions

- Summer '14 Midterm
  - 32-bit byte addressed MIPS
  - A two-way set associative 16KiB cache, a write-back policy, and 64B blocks
  - Assume a list of arrays containing 960 (i.e. 15*64 ints) elements

- The size of struct list is 64B

```
#define ARRAYNESS 15
typedef struct list {
  int vals[ARRAYNESS];
  struct list *next;
} list;
```

```
/* Version 1 */
void copy1(list *dst, list *src) {
  int i;
  while(src) {
    for (i=0; i<ARRAYNESS; i++)
      dst->vals[i] = src->vals[i];
    src = src->next;
    dst = dst->next;
  }
}
```

- Version1: Copy the whole array first, then move on to the next node in the linked list

```
/* Version 2 */
void copy2(list *dst, list *src) {
  int i;
  for (i=0; i<ARRAYNESS; i++) {
    lst *src2 = src, *dst2 = dst;
    while(src2) {
      dst2->vals[i] = src2->vals[i];
      src2 = src2->next;
      dst2 = dst2->next;
    }
  }
}
```

- Version2: Copy the i_{th} elements of each array through the linked list, then move on the $i+1_{th}$ elements of each array

(c) What is the best-case hit rate for version 1 of the code? 31/32 Version 2? 59/60

- In the best case,
  - dst == src
  - Every struct list fits in a cache block (block aligned)
  - No conflict misses
- Version1 – 1 compulsory miss per 32 accesses to each struct list
  - dst->vals[i] (15 writes), src->vals[i] (15 reads), src->next (1 read), dst->next (1 read) for each while iteration
- Version2 – 1 compulsory miss per struct list**, 64 misses in total** over **15 * 64 * 4 memory accesses in total**
  - 15 iteration for the for loop, 64 iterations for the while loop, 4 accesses for each while loop iteration, in total 15 * 64 * 4 accesses
  - Miss rate = 64 / (15*64*4) = 1/60, so hit rate = 59/60

Cache questions

- Summer '14 Midterm
  - 32-bit byte addressed MIPS
  - A two-way set associative 16KiB cache, a write-back policy, and 64B blocks
  - Assume a list of arrays containing 960 (i.e. 15*64 ints) elements

- The size of struct list is 64B

```
#define ARRAYNESS 15
typedef struct list {
  int vals[ARRAYNESS];
  struct list *next;
} list;
```

```
/* Version 1 */
void copy1(list *dst, list *src) {
  int i;
  while(src) {
    for (i=0; i<ARRAYNESS; i++)
      dst->vals[i] = src->vals[i];
    src = src->next;
    dst = dst->next;
  }
}
```

- Version1: Copy the whole array first, then move on to the next node in the linked list

```
/* Version 2 */
void copy2(list *dst, list *src) {
  int i;
  for (i=0; i<ARRAYNESS; i++) {
    lst *src2 = src, *dst2 = dst;
    while(src2) {
      dst2->vals[i] = src2->vals[i];
      src2 = src2->next;
      dst2 = dst2->next;
    }
  }
}
```

- Version2: Copy the i_{th} elements of each array through the linked list, then move on the $i+1_{th}$ elements of each array

(d) What is the worst-case hit rate for version 1 of the code? 7/8 Version 2? 0

- In the worst case,
  - dst != src
  - Every struct list appears in two cache blocks (not block aligned)
  - Every block maps to the same set (two-ways -> two rows per set)
- Version 1 – 4 compulsory misses (2 misses for both dst and src) per 32 accesses
- Version 2
  - All accesses are cache misses due to conflicts (two rows per set, two blocks per struct list)

Cache questions

- Summer '14 Midterm
  - 32-bit byte addressed MIPS
  - A two-way set associative 16KiB cache, a write-back policy, and 64B blocks
  - Assume a list of arrays containing 960 (i.e. 15*64 ints) elements

- The size of struct list is 64B

```
#define ARRAYNESS 15
typedef struct list {
  int vals[ARRAYNESS];
  struct list *next;
} list;
```

```
/* Version 1 */
void copy1(list *dst, list *src) {
  int i;
  while(src) {
    for (i=0; i<ARRAYNESS; i++)
      dst->vals[i] = src->vals[i];
    src = src->next;
    dst = dst->next;
  }
}
```

- Version1: Copy the whole array first, then move on to the next node in the linked list

```
/* Version 2 */
void copy2(list *dst, list *src) {
  int i;
  for (i=0; i<ARRAYNESS; i++) {
    lst *src2 = src, *dst2 = dst;
    while(src2) {
      dst2->vals[i] = src2->vals[i];
      src2 = src2->next;
      dst2 = dst2->next;
    }
  }
}
```

- Version2: Copy the i_{th} elements of each array through the linked list, then move on the $i+1_{th}$ elements of each array

(e) If our cache were 4-way set associative, what would the best case hit rate be for version 2? 59/60

- Same as (c), in the best case,
  - dst == src
  - Every struct list fits in a cache block (block aligned)
  - No conflict misses
- Version2 – 1 compulsory miss per struct list, **64 misses in total** over **15 * 64 * 4 memory accesses in total**
  - 15 iteration for the for loop, 64 iterations for the while loop, 4 accesses for each while loop iteration, in total 15 * 64 * 4 accesses
  - Miss rate = 64 / (15*64*4) = 1/60, so hit rate = 59/60

Cache questions

- Summer '14 Midterm
  - 32-bit byte addressed MIPS
  - A two-way set associative 16KiB cache, a write-back policy, and 64B blocks
  - Assume a list of arrays containing 960 (i.e. 15*64 ints) elements

- The size of struct list is 64B

```
#define ARRAYNESS 15
typedef struct list {
  int vals[ARRAYNESS];
  struct list *next;
} list;
```

```
/* Version 1 */
void copy1(list *dst, list *src) {
  int i;
  while(src) {
    for (i=0; i<ARRAYNESS; i++)
      dst->vals[i] = src->vals[i];
    src = src->next;
    dst = dst->next;
  }
}
```

- Version1: Copy the whole array first, then move on to the next node in the linked list

```
/* Version 2 */
void copy2(list *dst, list *src) {
  int i;
  for (i=0; i<ARRAYNESS; i++) {
    lst *src2 = src, *dst2 = dst;
    while(src2) {
      dst2->vals[i] = src2->vals[i];
      src2 = src2->next;
      dst2 = dst2->next;
    }
  }
}
```

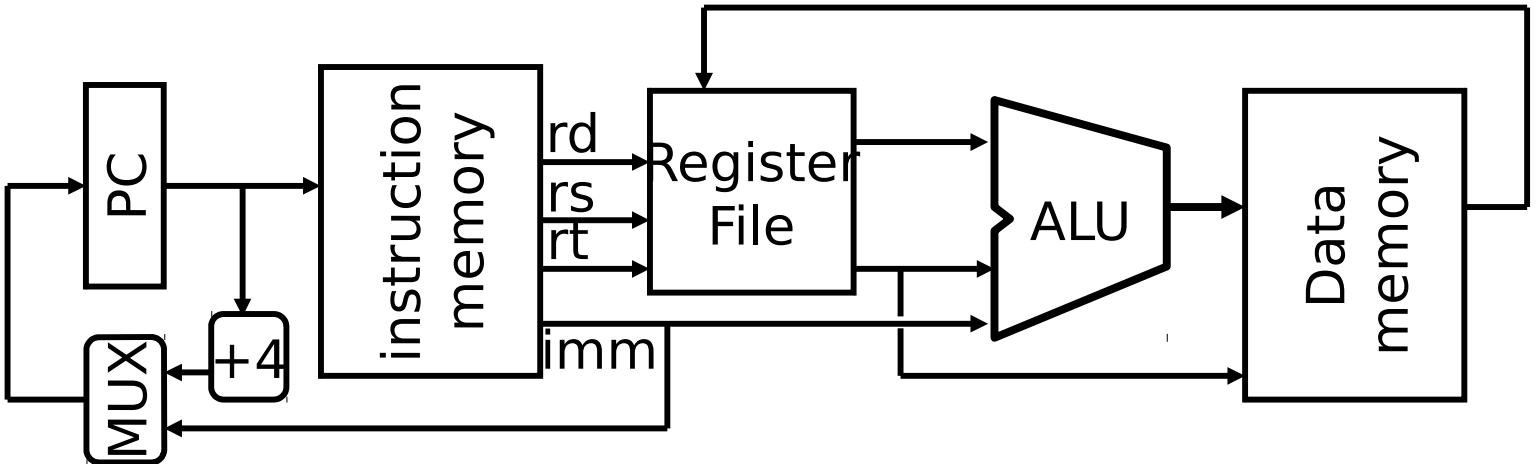
- Version2: Copy the i_{th} elements of each array through the linked list, then move on the $i+1_{th}$ elements of each array

(f) If our cache were fully associative, what would the worst case hit rate be for version 2? 14/15

- Similar to (d), in the worst case,
  - dst != src
  - Every struct list appears in two cache blocks (not block aligned)
  - Every block maps to the same set (two-ways -> two rows per set)
- Version2 – But, we don't have conflict misses since the cache is fully associative
  - 4 compulsory misses per struct list**, 64 * 4 misses in total** over **15 * 64 * 4 memory accesses in total**
  - 15 iteration for the for loop, 64 iterations for the while loop, 4 accesses for each while loop iteration, in total 15 * 64 * 4 accesses
  - Miss rate = (64 * 4) / (15*64*4) = 1/15, so hit rate = 14/15

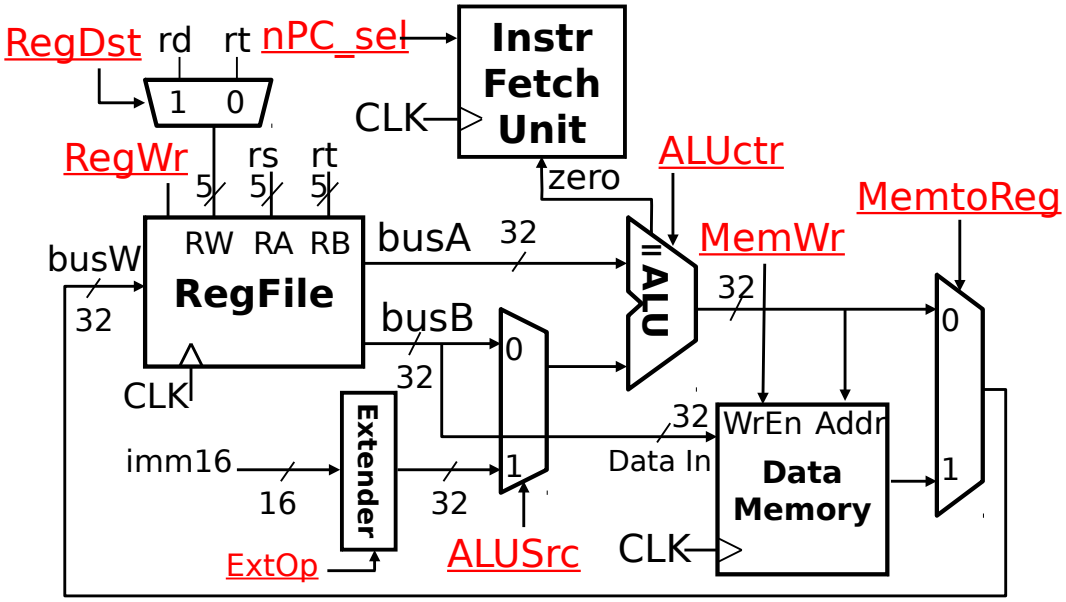
CPU Datapath and Control

- CPU Datapath
  - Part of the CPU
  - Hardware necessary to perform all operations
  - Includes
    - All registers (including PC, and those in the register file)
    - Functional units (adders, sign extenders, ALU)
    - Memory (e.g. instruction and data caches)



CPU Datapath and Control

- CPU control signals
  - Route parts of datapath: control signals for muxes
  - Control functional units: e.g. ALUctr, ExtOp
  - Specify write enable signals for storage elements (e.g. register file, memory)



CPU Datapath and Control

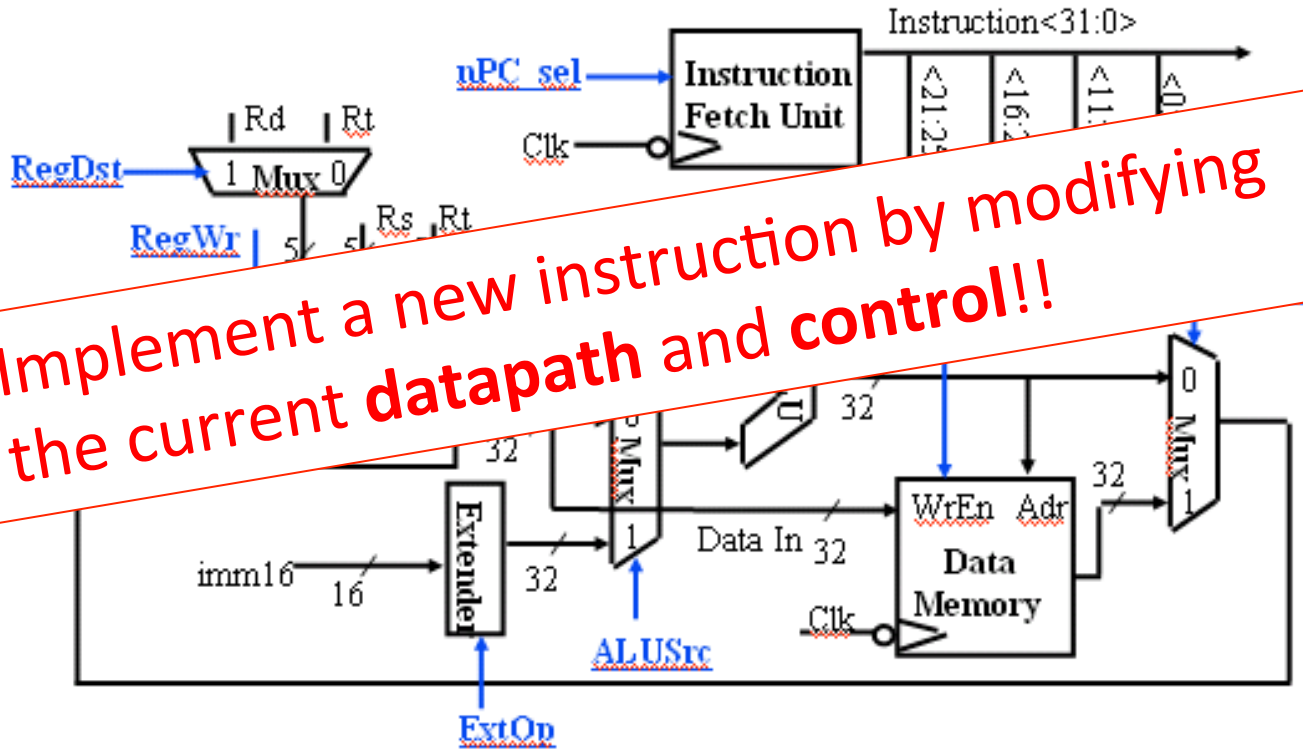
- Spring '07 final

Question F2: Control and Datapath (18 Points – 24 Minutes)

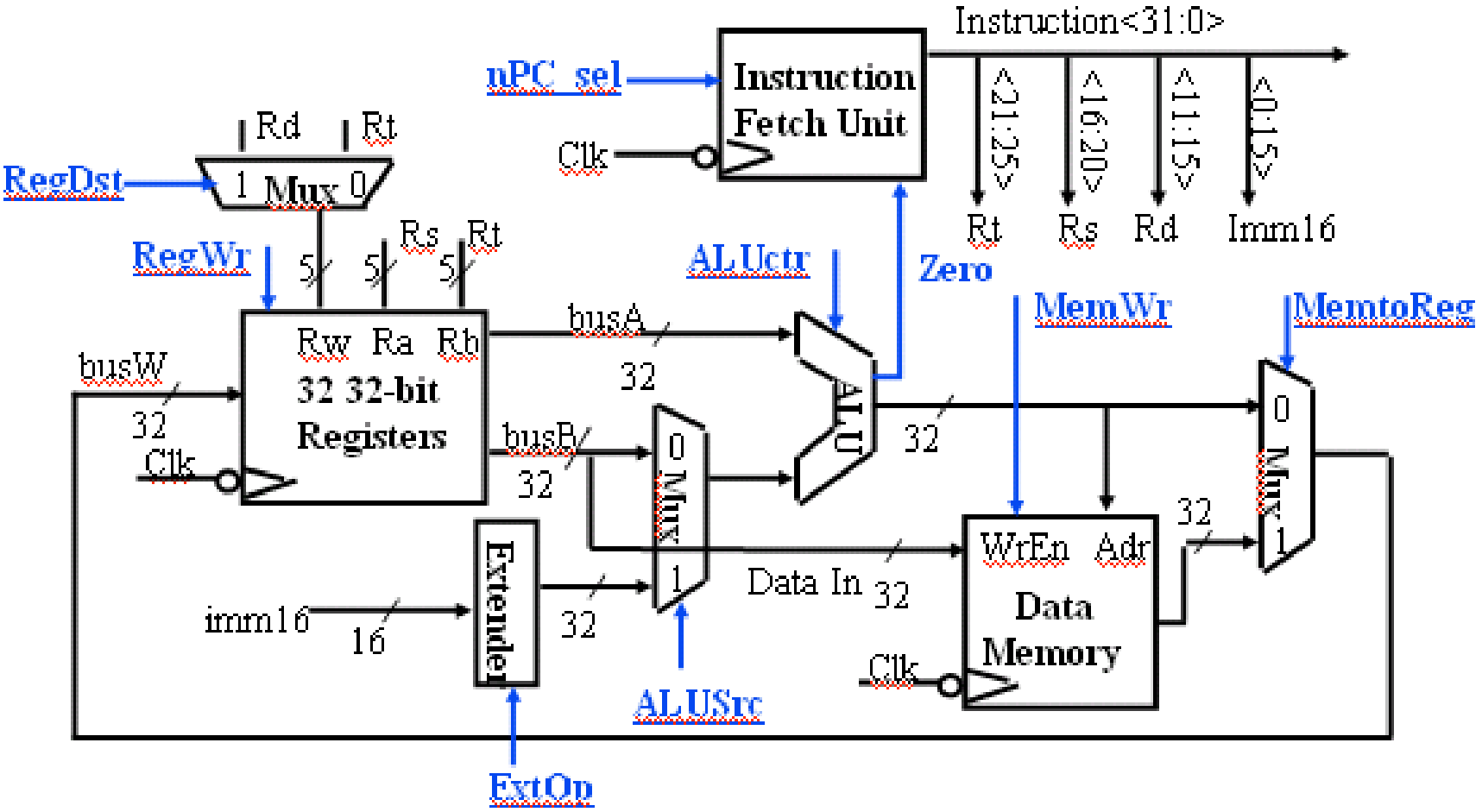
Modify the following single cycle MIPS datapath diagram to accomodate a new instruction `swai` (store word then auto-increment). The operation performs the regular `sw` operation, then post-increments the `rs` register by 1. Your modification may use simple adders, mux chips, wires, and new control signals. You may replace original labels where necessary. Recall the RTL for `sw` is:

`Mem[ R[rs] + SignExt[imm16] ] = R[rt]; PC=PC+4`, & that `sw` (and `swai`) has the following fields:

Opcode	Rs	Rt	Immediate



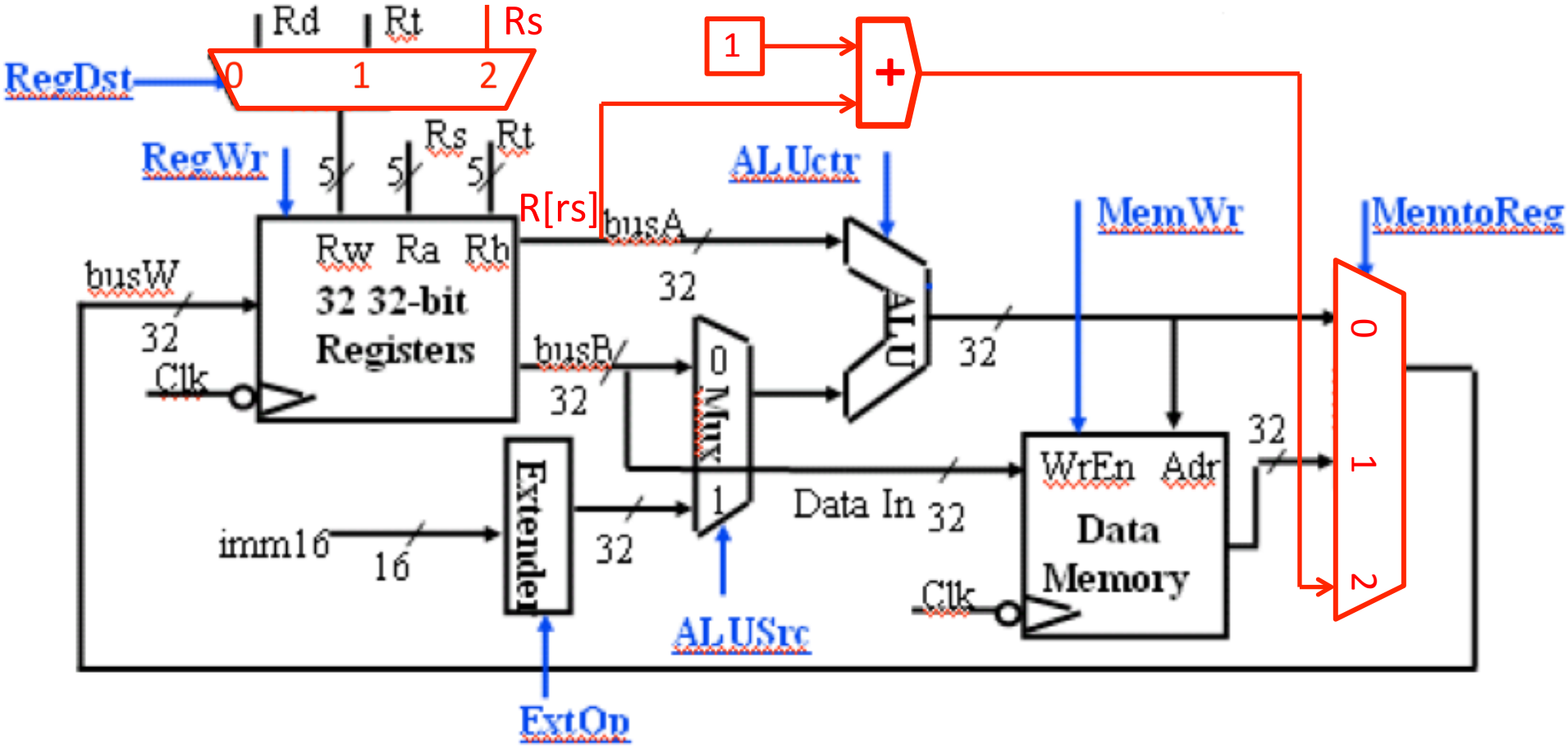
- Implement a new instruction by modifying the current **datapath** and **control**!!



- swai – store word, then increment rs by 1
- RTL (Register Transfer Language)
  - Mem[R[rs] + SignExt[imm16]] = R[rt]; (same as store)
  - R[rs] = R[rs] + 1

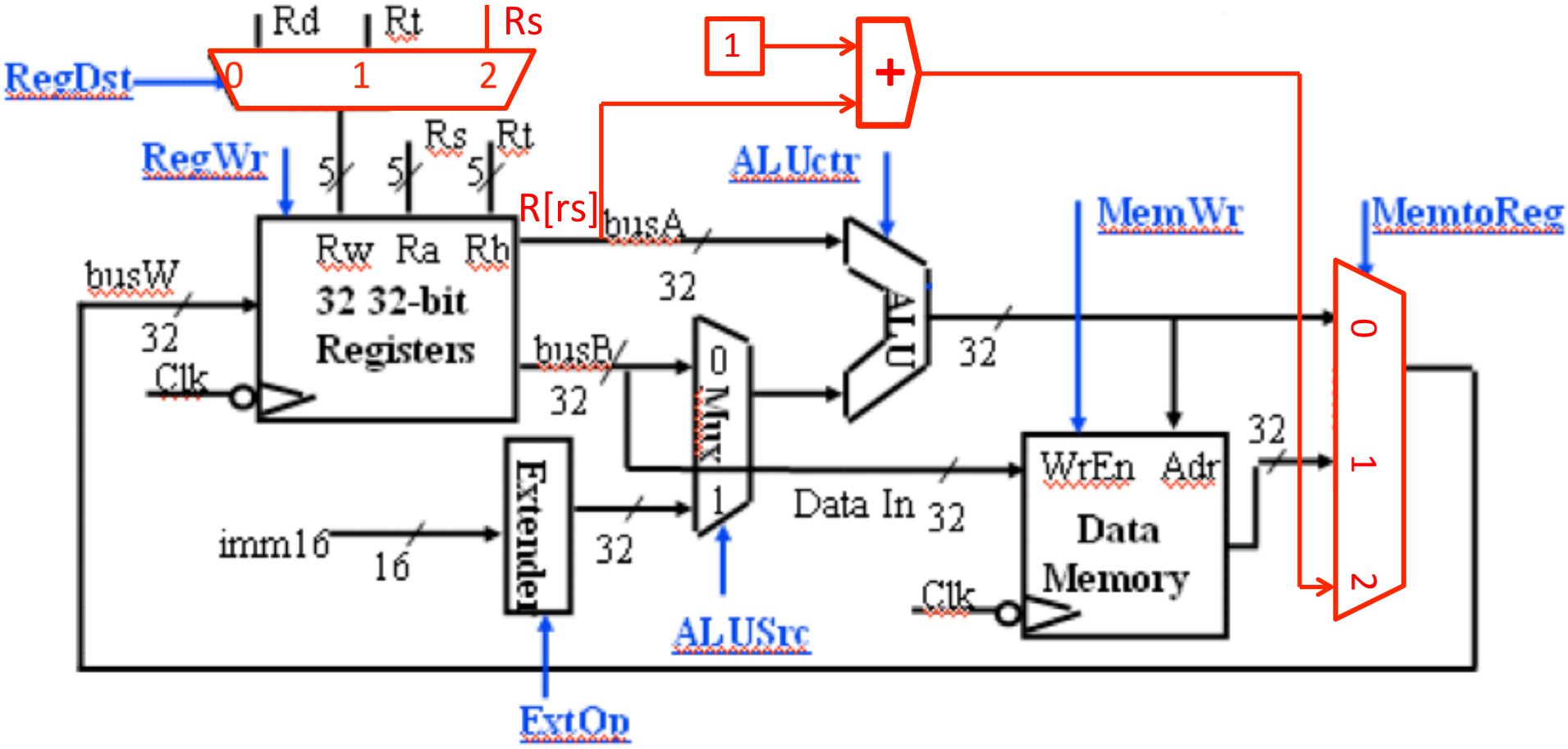
CPU Datapath and Control

- Modify datapath for R[rs] = R[rs] + 1

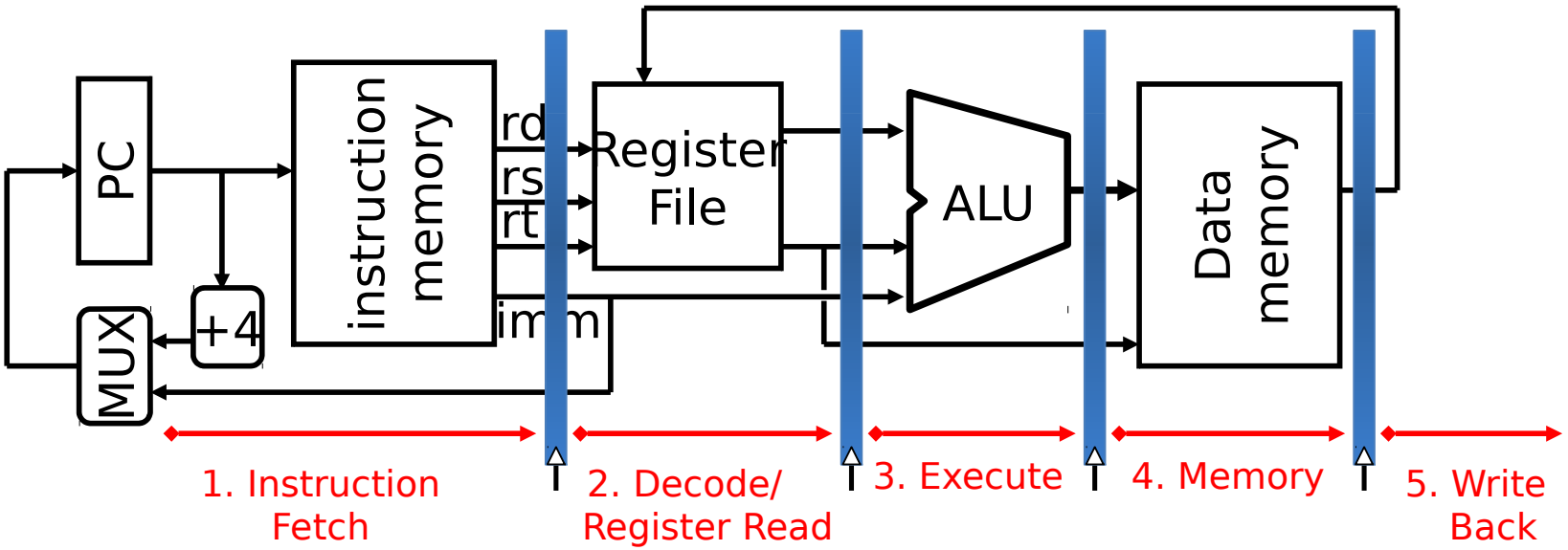


CPU Datapath and Control

- Modify datapath for R[rs] = R[rs] + 1
- Modify control signals for muxes
  - MemtoReg: width 1bit -> 2bits, selects R[rs]+1 for swai instruction
  - RegDst: width 1bit -> 2bits, selects Rs for swai instruction



CPU Pipelining Stages



- What happens in each pipeline stage?
- Registers?
  - Hold information produced in the previous cycle
- Main benefit of pipelining
  - Shorter critical path -> faster clock rate

Pipelining Hazards

- A hazard prevents starting the next instruction in the next cycle
- Type of hazards
  - Structural hazard
    - Resource is needed in multiple stages
  - Data hazard
    - Data dependency between instructions
  - Control hazard
    - Execution flow depends on previous instruction (e.g. branches and jumps)

CPU Pipelining questions

- Summer ’12 final

We are using a 5-stage MIPS pipelined datapath with separate I$ and D$ that can read and write to registers in a single cycle. Assume no other optimizations (no forwarding, etc.). The default behavior is to stall when necessary. Multiplication and branch checking are done during EX and the HI and LO registers are read during ID and written during WB.

a) As a reminder, T_c stands for “time between completions of instructions.” Given the following datapath stage times, what is the ratio $T_{c,\text{single-cycle}}/T_{c,\text{pipelined}}$? _____

IF	ID	EX	MEM	WB
200 ps	100 ps	400 ps	200 ps	100 ps

- $T_{c,\text{single-cycle}} = 200+100+400+200+100 = 1000$
- $T_{c,\text{pipelined}} = \text{Max}(200,100,400,200,100) = 400$
- $T_{c,\text{single-cycle}} / T_{c,\text{pipelined}} = 1000/400 = 5/2$

CPU Pipelining questions

- Summer '12 final

We are using a 5-stage MIPS pipelined datapath with separate I$ and D$ that can read and write to registers in a single cycle. Assume no other optimizations (no forwarding, etc.). The default behavior is to stall when necessary. Multiplication and branch checking are done during EX and the HI and LO registers are read during ID and written during WB.

```
        integrate:
1             beq   $a1,$0,exit
2             sub   $t0,$0,$a2    # $t0 = -alpha
3             addi  $t0,$t0,1     # $t0 = 1-alpha
4             lw    $t1,0($a0)    # $t1 = x[n]
5             mult  $t0,$t1
6             mflo  $t1           # $t1 = (1-alpha)*x[n]
7             sw    $t1,4($a0)    # x[n+1] = (1-alpha)*x[n]
8             addiu $a0,$a0,4
9             addiu $a1,$a1,-1
10            j     integrate
11    exit:   jr    $ra
```

b) Count the number of the different types of potential hazards found in the code above:

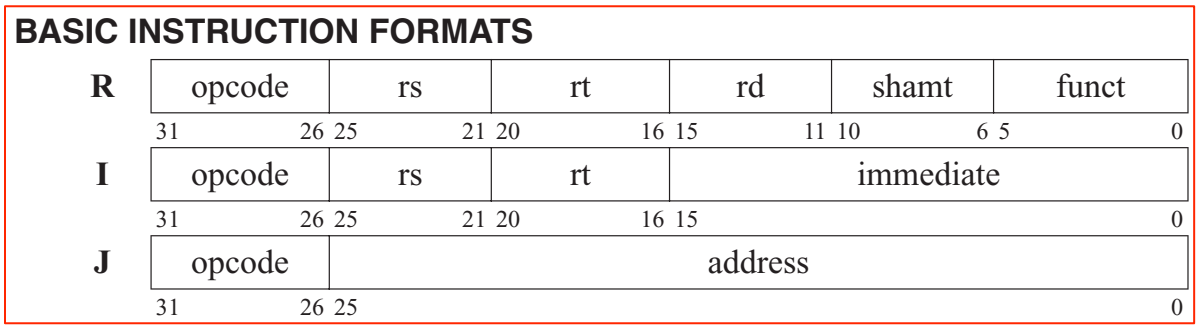
Structural: 0

Data: 6 @ 2-3, 3-5, 4-5, 5-6, 6-7, 9-1

Control: 3 @ 1, 10, 11

MIPS

- "MIPS Green Sheet" explains almost everything about MIPS ISA
  - Really need to get familiar with it!
- What's the maximum possible number of R format instructions MIPS can have?



(1) opcode(31:26) == 0

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary
(1)	`sll`	`add.f`	00 0000

  - For R format, opcode == 0, 2^6 possible functs
  - 2^6 = 64 possible R format instructions
- What's the maximum possible number of I and J format instructions MIPS can have?
  - For I & J format, opcode != 0, $2^6 - 1$ possible opcodes
  - $2^6 - 1$ = 63 possible I and J format instructions

MIPS Mystery Questions

- Summer '12 final

```
Mystery:
        la   $t0, L2
        lw   $t1, 8($t0)
        addi $t2, $0, 1
        sll  $t2, $t2, 16
        add  $t3, $0, $0
        add  $v0, $0, $0
        addi $t5, $0, 4
        sll  $t5, $t5, 16
L1:     beq  $t3, $t5, L3
        addu $t4, $t1, $t3
L2:     addu $t3, $t3, $t2
        sw   $t4, 8($t0)
        addu $v0, $v0, $a0
        j    L1
L3:     sw   $t1, 8($t0)
        jr   $ra
```

a) Which instruction gets modified during this function call?

1) $t0 holds address of L2 (the instruction "addu $t3, $t3, $t2")
2) Writing to address 8($t0)
3) 8($t0) points to the instruction "addu $v0, $v0, $a0"
4) **addu $v0, $v0, $a0 gets modified**

MIPS Mystery Questions

- Summer '12 final

```
Mystery:
        la   $t0, L2
        lw   $t1, 8($t0)
        addi $t2, $0, 1
        sll  $t2, $t2, 16
        add  $t3, $0, $0
        add  $v0, $0, $0
        addi $t5, $0, 4
        sll  $t5, $t5, 16
L1:     beq  $t3, $t5, L3
        addu $t4, $t1, $t3
L2:     addu $t3, $t3, $t2
        sw   $t4, 8($t0)
        addu $v0, $v0, $a0
        j    L1
L3:     sw   $t1, 8($t0)
        jr   $ra
```

b) How many times does the line at label `L2` get executed?

1) "L1: " and "j L1" form a loop, including L2
2) Loop is broken when $t3 == $t5
3) $t5 is 4<<16
4) $t3 gets incremented by $t2, which is 1<<16, every iteration
5) **L2 gets executed 4 times**

MIPS Mystery Questions

- Summer '12 final

```
Mystery:
        la   $t0, L2
        lw   $t1, 8($t0)
        addi $t2, $0, 1
        sll  $t2, $t2, 16
        add  $t3, $0, $0
        add  $v0, $0, $0
        addi $t5, $0, 4
        sll  $t5, $t5, 16
L1:     beq  $t3, $t5, L3
        addu $t4, $t1, $t3
L2:     addu $t3, $t3, $t2
        sw   $t4, 8($t0)
        addu $v0, $v0, $a0
        j    L1
L3:     sw   $t1, 8($t0)
        jr   $ra
```

c) Describe in a sentence or two what this function does.

1) To solve this Mystery, we need to know how does the instruction "addu $v0, $v0, $a0" get changed
2) $t3, which contains 1<<16, is added to the original instruction at each iteration
3) This increments $rt of addu, which is $a0, by one at each iteration
4) $rt of addu becomes $a0->$a1->$a2->$a3
5) **This Mystery returns the sum of $a0 ~ $a3**