ELE 455/555
Computer System Engineering

Section 1 – Review and Foundations

Class 6 – ISAs

ISA

MIPS

- This section follows the text very closely

ISA
MIPS

- The MIPS ISA contains 32 – 32bit registers
  - Registers $t0 - $t9 are used to hold temporary values
    - physical registers 8-15, 24, 25
  - Registers $s0 - $s9 are used to store saved values
    - typically variables from the original C code
    - physical registers 16-23
  - $zero – holds the constant value “0”
    - physical register 0
  - Others will be described later
- MIPS is a Load-Store architecture
  - operands are in registers for arithmetic and logical operations

ISA
MIPS

- Register Operations
  - Addition and Subtraction
  - Naturally atomic in nature
  - Simplest form is

    C = A + B C = A – B
  - MIPS always uses this simple format
    - Regularity → simpler implementation
    - Simplicity → higher performance and lower cost
  - Assembly:

```
add     c,a,b     # c gets a+b

add     $s2,$s0,$s1     #  s2 gets s0 + s1
```

ISA
MIPS

- Register Operations
  - C code:

    f = (g + h) - (i + j);

  - Compiled MIPS code:

```
add     t0, g, h        # temp t0 = g + h
add     t1, i, j        # temp t1 = i + j
sub     f, t0, t1       # f = t0 - t1

add     $t0, $s1, $s2       # temp $t0 = $s1 + $s2
add     $t1, $s3, $s4       # temp $t1 = $s3 + $s4
sub     $s0, $t0, $ t1      # $s0 = $t0 - $t1
```

ONLY USES REGISTERS

ISA
MIPS

- Memory Operations
  - Main memory used to store data
  - Arithmetic and Logical operations require data to be moved to/from main memory from/to a register
  - Words are 32 bits wide
    - Registers are 32 bits wide
    - Memory words are 32 bits wide
  - Memory is addressed in bytes
    - Each word is actually 4 bytes
    - Memory addresses are always aligned -> always a multiple of 4

→ PC increments by 4 after each fetch

ISA
MIPS

- Memory Operations
  - MIPS is “Big Endian”
    - Most significant byte is at least address of the word
    - e.g. 32 bit word 0x2FE34567

Memory Address (Hex)	Data Value
. . .	
1FFF	
2000	2F
2001	E3
2002	45
2003	67
2004	
. . .	

Memory Address (Hex)	Data Value
. . .	
2004	
2003	67
2002	45
2001	E3
2000	2F
1FFF	
. . .	

  - Some processors are “Little Endian”
    - Least significant byte is at least address of the word
    - e.g 32 bit word 0x2FE34567

Memory Address (Hex)	Data Value
. . .	
1FFF	
2000	67
2001	45
2002	E3
2003	2F
2004	
. . .	

Memory Address (Hex)	Data Value
. . .	
2004	
2003	2F
2002	E3
2001	45
2000	67
1FFF	
. . .	

ISA
MIPS

- Memory Operations

  - C code:

    g = h + A[8];

  - Compiled MIPS code:
    - assign g to $s1, h to $s2, base address of A to $s3
    - note: index of 8 means 8 words → 32 bytes

```
lw      $t0,32($s3)      # load word @ base register $s3 offset by 8 words
add     $s1,$s2,$t0
```

    - memory reference format is offset(base register)
    - With no designator, values are assumed to be decimal

ISA
MIPS

- Memory Operations

  - C code:
    A[12] = h + A[8];

  - Compiled MIPS code:
    - assign h to $s2, base address of A to $s3
    - note: index of 8 means 8 words → 32 bytes
    - note: index of 12 means 12 words → 48 bytes

```
lw    $t0,32($s3)    # load word @ base register $s3 offset by 8 words
add   $t0,$s2,$t0    # $t0 = $s2 + $t0
sw    $t0,48($s3)    # store word to base register $s3 offset by 12 words
```

    - memory reference format is offset(base register)
    - With no designator, values are assumed to be decimal

ISA
MIPS

- Memory Operations
  - MIPS memory configuration

Memory Address (Hex)	Data Value
. . .	
2004	12
2003	67
2002	45
2001	E3
2000	2F
1FFF	
. . .	

Memory Address (Hex)	Data Value
. . .	
2010	
200C	98F6 B352
2008	AB12 FE43
2004	1234 4321
2000	2FE3 4567
1FFC	
. . .	

ISA
MIPS

- Immediate Operands
  - Constants
  - Stored in program memory, not data memory
  - Avoid the need for a separate load instruction
  - Included in the 32 bit instruction field
    - No additional fetch required
    - Limited range

```
addi    $s3,$s3,4    # increments $s3 by 4 (address??)

addi    $s3,$s3,-4   # decrement $s3 by 4*
```

    - No subtract immediate needed

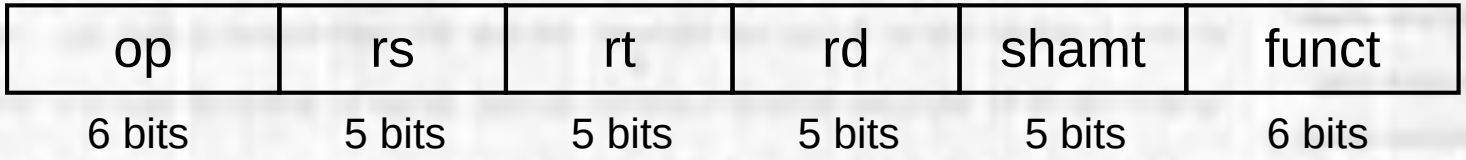
* in machine code this would be 2's compliment format
-4 → FFFC

ISA
MIPS

- Instruction Encoding
  - Machine Code
  - 32 bit word
  - 3 primary formats
    - Register
    - Immediate
    - Jump
  - 2 floating point formats
    - Floating Point Register
    - Floating Point Immediate
  - Simplified decoding → Small and Fast implementations

ISA
MIPS

- Instruction Encoding – R-format



- Instruction fields
  - op: operation code (opcode)
  - rs: first source register number
  - rt: second source register number
  - rd: destination register number
  - shamt: shift amount
  - funct: function code (extends opcode)

ISA
MIPS

- Instruction Encoding – R-format

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

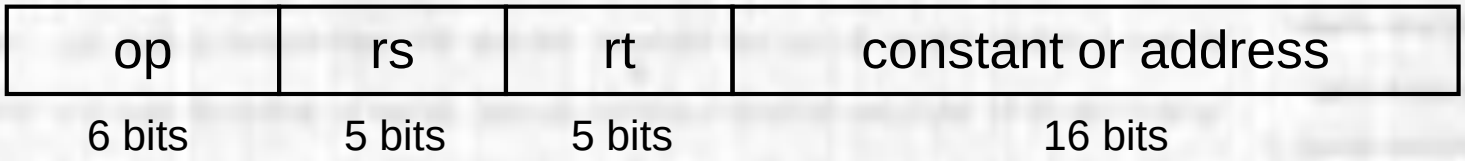
```
add $t0, $s1, $s2
```

add	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

$$00000010001100100100000000100000_2 = 02324020_{16}$$

ISA
MIPS

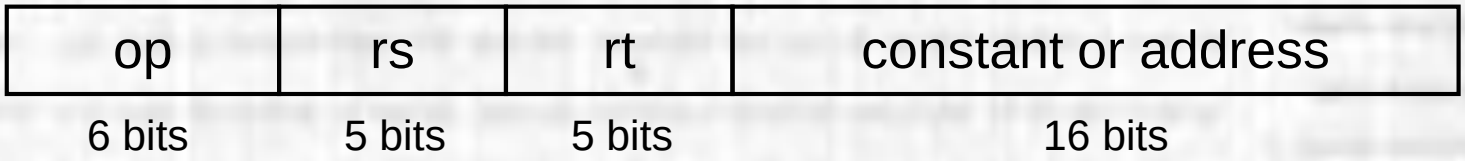
- Instruction Encoding – I-format



- Instruction Fields
  - op: operation code
  - Immediate arithmetic instructions
    - rs: source register number
    - rt: destination register number
    - constant: -2^{15} to $+2^{15} - 1$ (2's compliment)
  - Load/Store instructions
    - rs: register number with base address in it
    - rt: source or destination register number
    - address: offset added to base address in rs (2's compliment)

ISA
MIPS

- Instruction Encoding – I-format



- Instruction Fields
  - op: operation code
  - Branch instructions
    - rs: first test register number
    - rt: second test register number
    - PC relative addressing
      - address: offset (# of address words) to be added to PC
      - PC will already have advanced by 4
      - offset is multiplied by 4 in the processor to calculate the address
      - 2's compliment encoding

ISA
MIPS

- Instruction Encoding – I-format

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

```
addi    $t1,$t0,4    # increment $t0 by 4
```

addi	$t0	$t1	4
8	8	9	4
001000	01000	01001	0000000000000100

$0010\ 0001\ 0000\ 1001\ 0000\ 0000\ 0000\ 0100_2 = 2109\ 0004_{16}$

ISA
MIPS

- Instruction Encoding – I-format

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

```
lw   $t4,4($t0)    # load $t4 from memory location ($t0)+4
```

lw	$t0	$t4	4
35	8	12	4
100011	01000	01100	0000000000000100

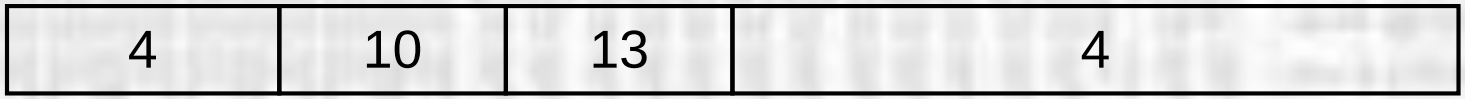
$1000\ 1101\ 0000\ 1100\ 0000\ 0000\ 0000\ 0100_2 = 8d0c\ 0004_{16}$

ISA
MIPS

- Instruction Encoding – I-format

6 bits 5 bits 5 bits 16 bits

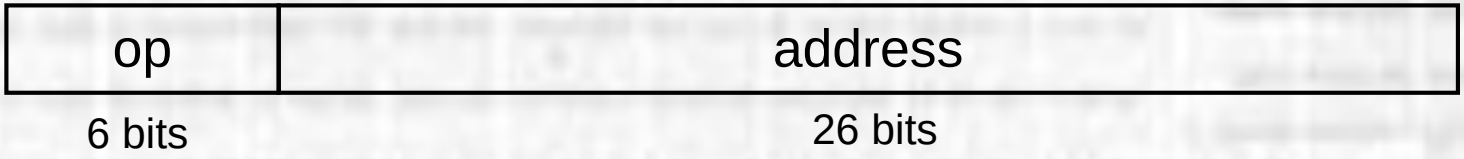
```
      beq   $t2,$t5,loop  # branch to loop if $t2 = $t5
      xyz
      xyz
      xyz
      xyz
loop  xyz
```



$0001\ 0001\ 0100\ 1101\ 0000\ 0000\ 0000\ 0100_2 = 114d\ 0004_{16}$

ISA
MIPS

- Instruction Encoding – J-format



- Instruction Fields
  - op: operation code
  - Direct jump addressing:
    - address: address to jump to
      - target address = address x 4
  - (Pseudo) Direct jump addressing
    - target address = $PC_{31\text{-}28}$: (address x 4)

ISA
MIPS

- Instruction Encoding – J-format

Address (hex)

00400054 j loop # jump to loop

00400058 xyz

0040005c xyz

00400060 loop xyz

j	4194400 (00400060_{16})
2	1048600 (100018_{16})
000010	00000100000000000000011000

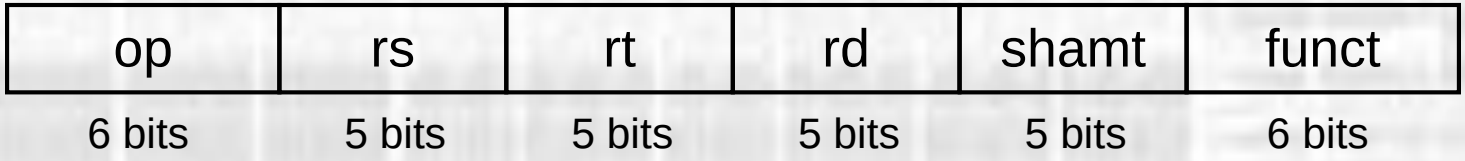
$0000\ 1000\ 0001\ 0000\ 0000\ 0000\ 0001\ 1000_2 = 0810\ 0018_{16}$

ISA
MIPS

- Logical Operations
  - Bitwise operations
  - R-format
    - sll – shift left logical
    - srl – shift right logical
    - and – bitwise logical AND (clear bits)
    - or – bitwise logical OR (set bits)
    - nor – bitwise logical NOT (invert bits)
  - I-format
    - andi – bitwise logical AND immediate
    - ori – bitwise logical OR immediate

ISA
MIPS

- Logical Operations
  - Shift operations



  - rs: unused
  - rt: - source register
  - rd: destination register
  - shamt: how many bits to shift - shift amount

  - fills with “0”s

ISA
MIPS

- Logical Operations - Shift

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

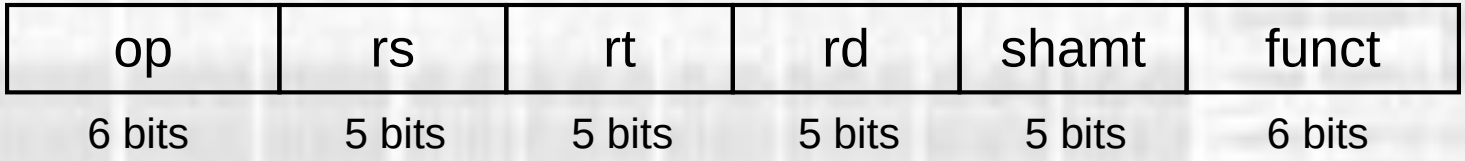
```
sll     $t2,$s0,4
```

sll	0	$s0	$t2	4	sll
0	0	16	10	4	0
000000	00000	10000	01010	00100	000000

$00000000000100000101000100000000_2 = 00105100_{16}$

ISA
MIPS

- Logical Operations
  - AND/OR/NOR operations



  - op: operation code (opcode)
  - rs: first source register number
  - rt: second source register number
  - rd: destination register number
  - shamt: shift amount – not used
  - funct: function code (extends opcode)

ISA
MIPS

- Logical Operations – AND, OR, NOR

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

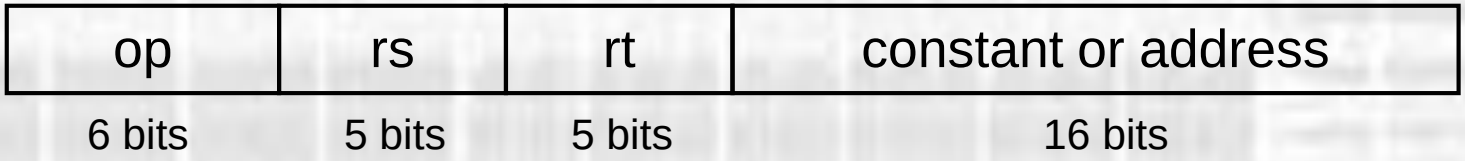
```
and     $t2,$s0,$s1
```

and	$s0	$s1	$t2	0	and
0	16	17	10	0	36
000000	10000	10001	01010	00000	100100

$00000010000100010101000000100100_2 = 02115024_{16}$

ISA
MIPS

- Logical Operations
  - AND/OR immediate



  - op: operation code
  - rs: source register number
  - rt: destination register number
  - constant: bits [15:0] for And/Or
    - Bits [31:16] are “0”

ISA
MIPS

- Logical Operations – AND, OR - immediate

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

```
andi    $t2,$s0,2048   # bits 31:16 = 0
```

andi	$s0	$t2	2048
12	16	10	2048
001100	10000	01010	0000100000000000

$0011\ 0010\ 0000\ 1010\ 0000\ 1000\ 0000\ 0000_2 = 320a0800_{16}$

ISA
MIPS

- Additional Instructions

- slt rd, rs, rt
  - Set if less than
  - if (rs < rt) rd = 1; else rd = 0;

- slti rt, rs, constant
  - Set if less than immediate
  - if (rs < constant) rt = 1; else rt = 0;

- Use in combination with beq, bne

```
slt $t0, $s1, $s2  # if ($s1 < $s2)
bne $t0, $zero, L  #   branch to L
```

Why not blt, bge, …?

ISA
MIPS

- Additional Instructions
  - Signed comparison: slt, slti
  - Unsigned comparison: sltu, sltui
  - Example
    - $s0 = 1111 1111 1111 1111 1111 1111 1111 1111
    - $s1 = 0000 0000 0000 0000 0000 0000 0000 0001

```
slt  $t0, $s0, $s1  # signed
```
–1 < +1 ⇒ $t0 = 1

```
sltu $t0, $s0, $s1  # unsigned
```
+4,294,967,295 > +1 ⇒ $t0 = 0

ISA
MIPS

- Example
  - C code:

```
if (i==j) f = g+h;
else f = g-h;
```

  - Compiled MIPS code:
    - Compiler assigns registers to variables: f, g, … in $s0, $s1, …

```
        bne $s3, $s4, else   # assembler replace label with offset
        add $s0, $s1, $s2
        j   exit             # assembler replace label with address
else:   sub $s0, $s1, $s2
exit:   …
```

ISA
MIPS

- Example
  - C code:

```
while (save[i] == k) i += 1;
```

  - Compiled MIPS code:
    - Compiler assigns registers to variables:
      - i in $s3, k in $s5, address of save in $s6

```
loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, exit    # assembler replace label with offset
      addi $s3, $s3, 1
      j    loop              # assembler replace label with address
exit: …
```

ISA
MIPS

- Summary
  - Load-Store Architecture
  - Small number of instructions
  - Small number of formats
    - R,I,J
  - Build complex operations from simple operations
    - blt, bge, …

→ Simplified structures
→ Fast operation

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $1,$2,$3	$1 = $2 + $3	3 operands; exception possible
	subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands; exception possible
	add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant; exception possible
	add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands; no exceptions
	subtract unsigned	subu $1,$2,$3	$1 = $2 – $3	3 operands; no exceptions
	add imm. unsign.	addiu $1,$2,100	$1 = $2 + 100	+ constant; no exceptions
	Move fr. copr. reg.	mfc0 $1,$epc	$1 = $epc	Used to get exception PC
	multiply	mult $2,$3	Hi, Lo = $2 ¥ $3	64-bit signed product in Hi, Lo
	multiply unsigned	multu $2,$3	Hi, Lo = $2 ¥ $3	64-bit unsigned product in Hi, Lo
	divide	div $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Lo = quotient, Hi = remainder
	divide unsigned	divu $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Unsigned quotient and remainder
	Move from Hi	mfhi $1	$1 = Hi	Used to get copy of Hi
	Move from Lo	mflo $1	$1 = Lo	Use to get copy of Lo
Logical	and	and $1,$2,$3	$1 = $2 & $3	3 register operands; logical AND
	or	or $1,$2,$3	$1 = $2 | $3	3 register operands; logical OR
	and immediate	andi $1,$2,100	$1 = $2 & 100	Logical AND register, constant
	or immediate	ori $1,$2,100	$1 = $2 | 100	Logical OR register, constant
	shift left logical	sll $1,$2,10	$1 = $2 << 10	Shift left by constant
	shift right logical	srl $1,$2,10	$1 = $2 >> 10	Shift right by constant
Data transfer	load word	lw $1,100($2)	$1 = Memory[$2+100]	Data from memory to register
	store word	sw $1,100($2)	Memory[$2+100] = $1	Data from register to memory
	load upper imm.	lui $1,100	$1 = 100 x 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq $1,$2,100	if ($1 == $2) go to PC+4+100	Equal test; PC relative branch
	branch on not eq.	bne $1,$2,100	if ($1 != $2) go to PC+4+100	Not equal test; PC relative
	set on less than	slt $1,$2,$3	if ($2 < $3) $1=1; else $1=0	Compare less than; 2's complement
	set less than imm.	slti $1,$2,100	if ($2 < 100) $1=1; else $1=0	Compare < constant; 2's comp.
	set less than uns.	sltu $1,$2,$3	if ($2 < $3) $1=1; else $1=0	Compare less than; natural number
	set l.t. imm. uns.	sltiu $1,$2,100	if ($2 < 100) $1=1; else $1=0	Compare < constant; natural
Unconditional jump	jump	j 10000	go to 10000	Jump to target address
	jump register	jr $31	go to $31	For switch, procedure return
	jump and link	jal 10000	$31 = PC + 4; go to 10000	For procedure call

ISA
MIPS

- Procedure Calling
  - Steps

1. Place parameters in registers
2. Transfer control to procedure
3. Acquire storage for procedure
4. Perform procedure's operations
5. Place result in register for caller
6. Return to place of call

ISA
MIPS

- Procedure Calling
  - Register Usage

		registers
• $a0 – $a3	arguments	4-7
• $v0, $v1	result values	2-3
• $t0 – $t9	temporaries	8-15,24,25
• Can be overwritten by callee		
• $s0 – $s7	saved	16-23
• Must be saved/restored by callee		
• $gp	global pointer for static data	28
• $sp	stack pointer	29
• $fp	frame pointer	30
• $ra	return address	31

ISA
MIPS

- Procedure Calling – leaf procedure
  - jump and link

```
jal ProcedureLabel
```

    - Address of following instruction put in $ra
    - Jumps to target address
  - jump register

```
jr $ra
```

    - Copies $ra to program counter
    - Can also be used for computed jumps
      - e.g., for case/switch statements

ISA

MIPS

- Procedure Calling – leaf procedure
  - C code:

```
int pcall (int g, h, i, j) # args g-j in $a0-$a3
{ int f;                   #  f in $s0 (hence, need to save $s0 on stack)
  f = (g + h) - (i + j);
  return f;                # result in $v0
}
```

  - MIPS code

```
pcall:   addi $sp, $sp, -4       # save $s0 on stack
         sw   $s0, 0($sp)        #      "
         add  $t0, $a0, $a1      # -----
         add  $t1, $a2, $a3      # procedure body
         sub  $s0, $t0, $t1      # -----
         add  $v0, $s0, $zero    # result
         lw   $s0, 0($sp)        # restore $s0
         addi $sp, $sp, 4        #      "
         jr   $ra                # return
```

ISA
MIPS

- Procedure Calling – non-leaf procedure
  - Procedures that call other procedures
    - Caller needs to save (on the stack)
      - Return address
      - Arguments needed after the call
      - Temporaries needed after the call
    - Restore from the stack on return

ISA
MIPS

- Procedure Calling – non-leaf procedure
  - C code

```
int fact (int n)
{
   if (n < 1) return f;
   else return n * fact(n - 1);
}
```

ISA

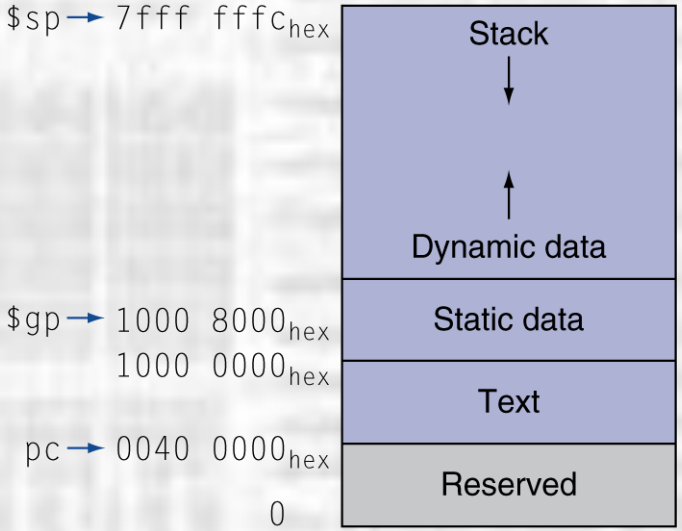
MIPS

- Procedure Calling – non-leaf procedure
  - MIPS code

```
fact:   addi $sp, $sp, -8       # adjust stack for 2 items
        sw   $ra, 4($sp)        # save return address
        sw   $a0, 0($sp)        # save argument
        slti $t0, $a0, 1        # test for n < 1
        beq  $t0, $zero, L1
        addi $v0, $zero, 1      # if so, result is 1
        addi $sp, $sp, 8        #   pop 2 items from stack
        jr   $ra                #   and return
L1:     addi $a0, $a0, -1       # else decrement n
        jal  fact               # recursive call
        lw   $a0, 0($sp)        # restore original n
        lw   $ra, 4($sp)        #   and return address
        addi $sp, $sp, 8        # pop 2 items from stack
        mul  $v0, $a0, $v0      # multiply to get result
        jr   $ra                # and return
```

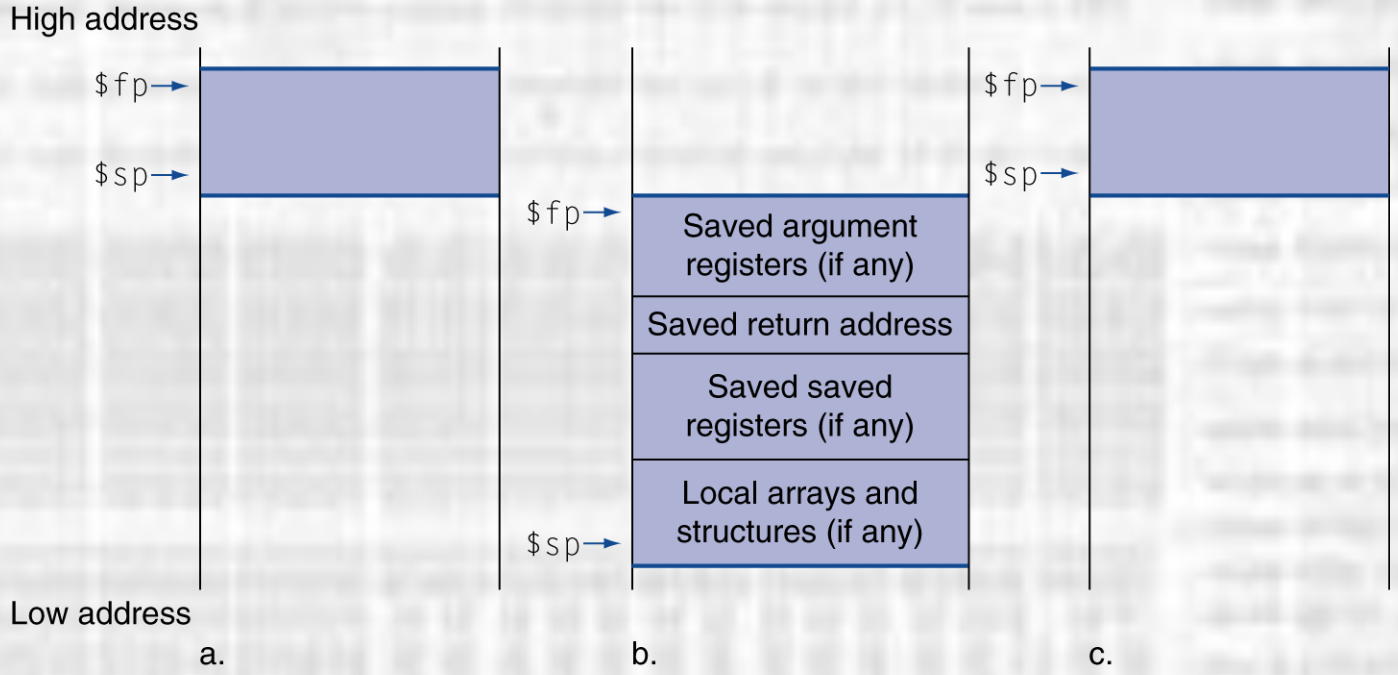
ISA
MIPS

- Memory Conventions
  - **Text:** program code
  - **Static data:** global variables
    - e.g., static variables in C, constant arrays and strings
    - $gp initialized to address allowing ±offsets into this segment
  - **Dynamic data:** heap
    - E.g., malloc in C, new in Java
  - **Stack:** automatic storage



ISA
MIPS

- Memory Conventions



- Local data allocated by callee
  - e.g., C automatic variables
- Procedure frame (activation record)
  - Used by some compilers to manage stack storage

ISA
ARM

- MIPS – ARM Comparison

	ARM	MIPS
Date announced	1985	1985
Instruction size	32 bits	32 bits
Address space	32-bit flat	32-bit flat
Data alignment	Aligned	Aligned
Data addressing modes	9	3
Registers	15 × 32-bit	31 × 32-bit
Input/output	Memory mapped	Memory mapped

ISA
ARM

- MIPS – ARM Comparison

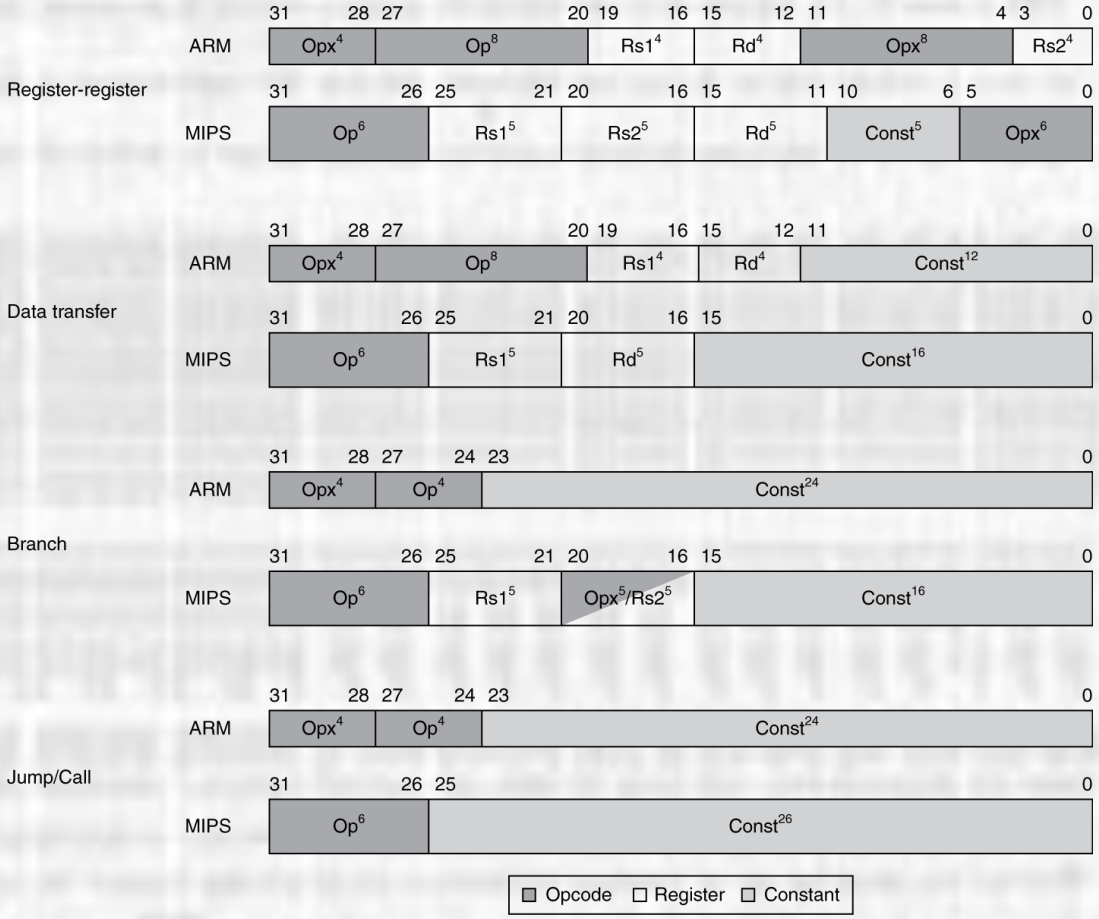
- ARM

  - Uses condition codes for result of an arithmetic/logical instruction
    - Negative, zero, carry, overflow
    - Compare instructions to set condition codes without keeping the result

  - Each instruction can be conditional
    - Top 4 bits of instruction word: condition value
    - Can avoid branches over single instructions

ISA
ARM

- MIPS – ARM Comparison



ISA
Benchmarking

- MIPS Relative Instruction Execution Rates

Instruction class	MIPS examples	SPEC2006 Int	SPEC2006 FP
Arithmetic	`add, sub, addi`	16%	48%
Data transfer	`lw, sw, lb, lbu, lh, lhu, sb, lui`	35%	36%
Logical	`and, or, nor, andi, ori, sll, srl`	12%	4%
Cond. Branch	`beq, bne, slt, slti, sltiu`	34%	8%
Jump	`j, jr, jal`	2%	0%