CS31001 COMPUTER ORGANIZATION AND ARCHITECTURE

Debdeep Mukhopadhyay,
CSE, IIT Kharagpur

Instructions and Addressing

ISA vs. Microarchitecture

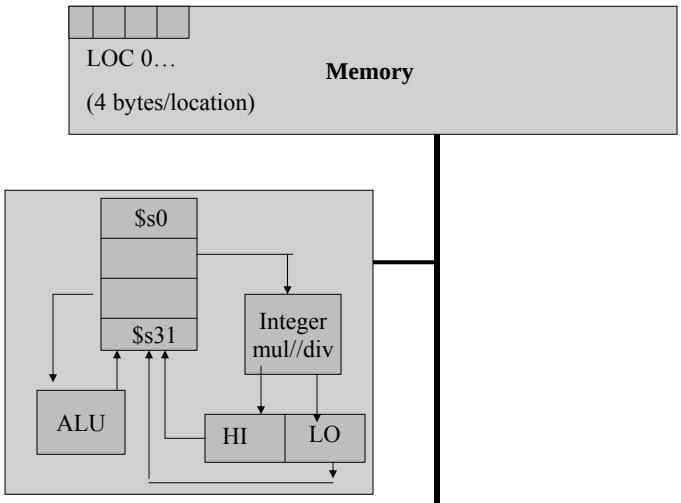
- An ISA or **Instruction Set Architecture** describes the aspects of a computer architecture visible to the low-level programmer, including the native datatypes, instructions, registers, addressing modes, memory architecture, interrupt and exception handling, and I/O organization.
  - ISA is a logical address.
- **Microarchitecture** is the set of internal processor design techniques used to implement the instruction set (including microcode, pipelining, cache systems etc.)

MIPS: Background

- MIPS: *Microprocessor without Interlocked Pipelined Stages*
- 1981: A Stanford University engineering team headed by Dr. John Hennessy initiates the MIPS RISC architecture project.
- 1984: MIPS Computer Systems, Inc. founded by Dr. John Hennessy.
- MIPS is a RISC microprocessor architecture developed by MIPS Technologies.
- 32-bit processor R3000 was developed in 1988 and the first 64-bit processor released in 1991.

A MIPS R4400 processor made by Toshiba.

Computer Organization



Registers and data sizes in MIPS

Symbolic Name	Number	Usage
zero	0	Constant 0.
at	1	Reserved for the assembler.
v0 - v1	2 - 3	Result Registers.
a0 - a3	4 - 7	Argument Registers $1 \cdots 4$.
t0 - t9	8 - 15, 24 - 25	Temporary Registers $0 \cdots 9$.
s0 - s7	16 - 23	Saved Registers $0 \cdots 7$.
k0 - k1	26 - 27	Kernel Registers $0 \cdots 1$.
gp	28	Global Data Pointer.
sp	29	Stack Pointer.
fp	30	Frame Pointer.
ra	31	Return Address.

Big Endian-Little Endian

- An important aspect is how the bytes in memory are indexed.
- Convention is right-most bit is assigned the index 0, and the left most bit is assigned the bit 31.

Byte 3	Byte 2	Byte 1	Byte 0

31 23 7 0

Words are stored as individually addressable bytes in memory M.

What is the storage order of the bytes?

Consider, a sequence of words, $W_0, W_1, \ldots W_m$ of (m+1) 4 byte words.

Suppose, $W_i = B_{i,3}, B_{i,2}, B_{i,1}, B_{i,0}$.

Thus, the sequence is:

$B_{0,3}, B_{0,2}, B_{0,1}, B_{0,0}, \ldots, B_{m,3}, B_{m,2}, B_{m,1}, B_{m,0}$

Two forms of addressing is available:

Big Endian: adr0,adr1,…adr(4m+3),

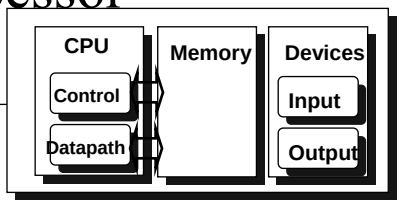
in increasing order.

[most significant byte is given the lowest address]

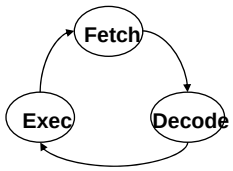
Little Endian:

$B_{0,0}, B_{0,1}, B_{0,2}, B_{0,3}, \ldots, B_{m,0}, B_{m,1}, B_{m,2}, B_{m,3}$

(von Neumann) Processor Organization



- **Control** needs to
  1. input instructions from Memory
  2. issue signals to control the information flow between the Datapath components and to control what operations they perform
  3. control instruction sequencing



- **Datapath** needs to have the
  - components – the functional units and storage (e.g., register file) needed to execute instructions
  - interconnects - components connected so that the instructions can be accomplished and so that data can be loaded from and stored to Memory

MIPS Arithmetic Instructions

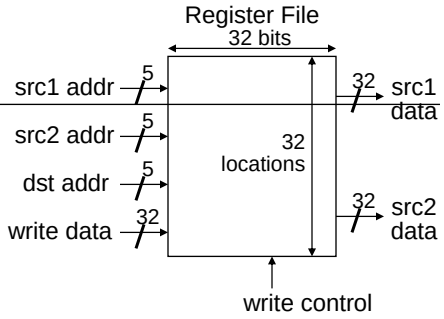
- MIPS assembly language arithmetic statement

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

- Each arithmetic instruction performs only one operation
- Each arithmetic instruction fits in 32 bits and specifies exactly three operands

  destination ← source1 op source2
- Operand order is fixed (destination first)
- Those operands are all contained in the datapath's register file (`$t0,$s1,$s2`) – indicated by $

MIPS Register File



- Holds thirty-two 32-bit registers
  - Two read ports and
  - One write port
- Registers are
  - Faster than main memory
    - But register files with more locations are slower (e.g., a 64 word file could be as much as 50% slower than a 32 word file)
    - Read/write port increase impacts speed quadratically
  - Easier for a compiler to use
    - e.g., (A*B) – (C*D) – (E*F) can do multiplies in any order vs. stack
  - Can hold variables so that
    - code density improves (since register are named with fewer bits than a memory location)

Machine Language - Add Instruction

- Instructions, like registers and words of data, are 32 bits long
- Arithmetic Instruction Format (**R** format):

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

op	rs	rt	rd	shamt	funct
000000	00111	01000	01100	00000	010100

op	6-bits	opcode that specifies the operation
rs	5-bits	register file address of the first source operand
rt	5-bits	register file address of the second source operand
rd	5-bits	register file address of the result's destination
shamt	5-bits	shift amount (for shift instructions)
funct	6-bits	function code augmenting the opcode

MIPS Memory Access Instructions

- MIPS has two basic data transfer instructions for accessing memory

```
lw $t0, 4($s3)  #load word from memory
sw $t0, 8($s3)  #store word to memory
```

- The data is loaded into (lw) or stored from (sw) a register in the register file – a 5 bit address
- The memory address – a 32 bit address – is formed by adding the contents of the base address register to the offset value
  - A 16-bit field meaning access is limited to memory locations within a region of $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of the address in the base register
  - Note that the offset can be positive or negative

Compiling using a variable Index

- g=h+A[i]
- Assume A is an array of 100 elements whose base is in register $s3 and the compiler associates the variables g, h, and i with the registers $s1, $s2 and $s4. What is the MIPS assembly code?
  - add $t1,$s4,$s4
  - add $t1,$t1,$t1
  - add $t1,$t1,$s3 #address of A[100]
  - lw $t0,0($t1)
  - add $s1,$s2,$t0

I-type Instructions

- R-type Instruction

6 bits	5 bits	5 bits	6 bits	5 bits	6 bits
OpCode	Source register 1	Source register 2	Destination register	Shift amount	function

- I-type (Immediate) Instructions

6 bits	5 bits	5 bits	16 bits
OpCode	Source register	Destination register 1	offset

Immediate and operations

- addi $t0,$s0,61

	rs (source)	rt (destination)	
001000	01000	10000	0000000000111101

Instructions are such that the constant can be directly added.

Adders in MIPS are 32 bits. So, the 16 bits are sign extended.

Other examples are: andi, ori, xori.

andi can be used to extract fields from a word.

Load and Store Instructions

opcode	base reg	data reg	offset (16 bit signed value)

- lw $t0,40($s3) #load mem[40+($s3)] into $t0
- sw $t0,A($s3) #store $t0 into mem[40+($s3)]

Instruction:

10x011	10011	01000	offset (16 bit signed value)
lw=35 sw=43	Base Reg	Data Reg	Offset relative to base

Loading a Constant

- addi $t0,$zero,constant #works if constant is
  #lesser then 16 bits
- For larger than 16 bits. use "lui" (load upper immediate) instruction:

  lui $s0, 61 #immediate value of 61 (decimal) is #loaded in the upper half of $s0, with the lower 16 #bits set to 0s.

	rs (source)	rt (destination)	
001111	00000	10000	0000000000111101

For the lower 16 bits

- Use the instruction "ori" (or-immediate)
  - Say we want to load constant "0x2110 003d" to $t0.
    - lui $t0,0x2110
    - ori $t0,0x003d
- How do you load the constant 0xffff ffff?
  - You can change the immediate operand.
  - Or, use the "nor" instruction.
    - nor $s0,$zero,$zero

Obtaining the Machine Code

- A[300]=h+A[300]
  - assume that $t1 has the base address of the array A and $s2 stores the value of h
  - opcodes for lw: 35, add:0, sw: 43
- Assembly:
  - lw $t0,1200($t1)
  - add $t0,$s2,$t0
  - sw $t0,1200($t1)
- Write the machine language instructions?

Jump and Branch Instructions

- Unconditional Jumps:
  - j endloop #go to memory loc "endloop"
  - jr $ra #go to location whose memoy address

    #is in $ra. $ra may hold the return address from

    #a procedure.
- The first instruction is a simple jump, which causes program execution to proceed from the location whose numeric or symbolic address is provided.
- The second one is called "jump register", specifies a register to hold the jump target address.
  - $ra, the register, is used to effect a return from a procedure to the point from which the procedure was called.

Instruction Formats

op	Jump Target Address
31 26	25 0

- For the j instructions, the 26-bit address field in the instruction, is augmented with:
  - 00 to the right
  - 4 higher order bits of the program counter to the left
- Called as Pseudodirect-addressing.

The j instructions in MIPS

op	Jump Target Address	
000010	25 0	
xxxx	25 0	00

The jr instructions in MIPS

- R-type

000000	11111	00000	00000	00000	001000
OpCode	Source register ($ra)	Unused	Unused	Unused	function jr=8

Conditional Branches

- These instructions allow us to transfer control to a given address when a condition is met.
- Conditions in MIPS ISA can be:
  - Register Content being negative
  - Equality of two register contents
  - Inequality of two register contents
- For the other kind of branchings, MIPS offers an R-type instruction, called as slt (set less than).
  - If “less than relationship” holds between two registers, a specified destination register is set to 1, else 0.

The Branch Statements in Assembly

- bltz $s1,L #branch to the symbolic memory L
- #if content of $s1<0
- beq $s1, $s2, L
- bne $s1, $s2, L
- slt $s1,$s2,$s3 #if the content of $s2<content #of $s3, set content of $s1 to #1, else 0.
- slti $s1, $s2, 61

Machine Language (M/L) Formats

op	rs (source)	rt (destination)	offset
000001	10001	00000	0000000000111101
bltz=1	**Source $s1**		**Relative Branch Distance in words**

op	rs (source)	rt (destination)	offset
00010x	10001	10010	0000000000111101
beq=4			**Relative Branch Distance in words**
bne=5			

Examples of PC relative addressing: the 16-bit signed offset is multiplied by 4 (why?) and added to the 32 bit PC, to get a 32 bit branch target address.

Other Branch Statements in M/L

- **slt:**

slt $s1,$s2,$s3

000000	10010	10011	10001	00000	101010
OpCode	Source register ($s2)	Source register ($s3)	Destination Register $s1	Unused	function slt=42

- **slti:**

001010	10010	10001	0000000000111101
OpCode slti=10	Source register ($s2)	Destination Register ($s1)	Immediate Operand

slti $s1,$s2,61

A finer point

- What if the label specified in a beq statement is too far to be reached via a 16-bit offset?
  - The assembler automatically replaces:

```
        beq $s0,$s1,L1
with:
        bne $s0,$s1,L2
        j L1
    L2: …
```

Assignment

- Write the assembly language code snippets for:
  - if(i==j) x=x+y;
  - if(i<j) x=x+y;
  - if(i<=j) {x=x+1; z=1;} else {y=y-1; z=z*2;}
  - while(A[i]==k) i=i+1;
  - loop: i=i+step;

    sum=sum+A[i];

    if(i≠n) goto loop;

Addressing Modes

- Methods by which the location of an operand is specified within an instruction:
  1. **Implied Addressing:** Operand comes from, or result goes to, a predefined place that is not explicitly defined in the instruction.

     Example: jal, address of next instruction is stored in $ra.
  2. **Immediate Addressing:** Operand is given in the instruction itself..

     Example: andi, ori, addi

Addressing Modes

3. **Register Addressing:** Operand is taken from, or result placed in, a specified register.

   Example: R-type Instructions.

4. **Base Addressing:** Operand is in memory and its location is computed by adding a 16-bit signed integer, the offset, with the contents of the base register, specified in the instruction.

   Example: lw, sw

Addressing Modes

5. **PC-relative addressing:** Same as base addressing, but the register is always the Program Counter (PC).

   Example: beq, bne

6. **Pseudo-direct addressing:** In direct addressing, the operand address is part of the instruction. However this is not possible in MIPS, as we have 32 bit instruction, and address also of 32 bits. Hence, we have pseudo-direct addressing.

   Example: j instruction

MIPS Instructions

- Please refer text book for the list.

Summary

- MIPS has a load/store architecture => operands must be in registers before they are executed.
- MIPS instructions for accessing memory: load, store, jump/branch
- MIPS has limited addressing modes:
  - efficient hardware design is possible.
  - Adequate for programming.