Chapter 1 Computer Abstractions and Tech

1.3 Under the Covers

- Opening the Box

* Motherboard- plastic board containing packages of integrated circuits or chips,
* Integrated circuit- called a chip. Combines dozens of transistors
* Dynamic Random Access Memory- memory built as an integrated circuit.
* DIMM- dual inline memory module- small board that contains DRAM chips on both sides
* Several are used together to contain instructions and data of a program
* CPU/Processor- active part, follows instructions of program.
* Datapath- component of processor that performs arithmetic ops
* Control- commands datapath, memory, and I/O devices according to instructions of program
* Cache memory- small, fast memory that acts as a buffer for the DRAM memory
* Static random access memory- memory built as an integrated circuit, but faster and less dense than DRAM
* Abstractions- model that renders low level details of computer system temporarily invisible to facilitate
* Instruction set architecture- interface between hardware and lower level software that encompasses all info necessary to write a machine language program that will run correctly
* Application binary interface- user portion of the instruction plus OS interfaces used by application programmers. Defines a standard for binary portability
* Implementation- hardware that obeys the architecture abstraction
* Volatile- when loses power, it forgets
* Main memory- hold programs while they are running consists of DRAM
* Three differences between magnetic disks and main memory
  + Disks are nonvolatile, have a slower access time, and cheaper per gig
* Flash memory- nonvolatile, 100-1000 times faster than disk, in cameras and music players
* Tech for Building Processors and Memory

Year Technology used in Tech used relative performance/ unit cost performance/unit cost

1951 Vacuum tube 0,000,001

1965 Transistor 0,000,035

1975 Integrated circuit 0,000,900

1995 Very large-scale integrated circuit 2,400,000

2005 Ultra large-scale integrated circuit 6,200,000,000

* + - Vacuum tube- electronic component, predecessor of the transistor, consists of a hollow glass tube about 5-10 cm long
    - Transistor- on/off switch controlled by an electric signal
    - Very large scale integrated circuit- device containing hundreds of thousands to millions of resistors

1.4 Performance

* Defining Performance
  + - Throughput or bandwidth- total amount of work done in a given time
      * Decrease response time improves throughput
        + EX 1 replace processor for faster one. – increases response and throughput
        + EX 2 add additional processors – only throughput increases
    - TO max performance, we want to min response time or execution time
    - Performance = 1/ execution time
    - Performance(x)/ Performance(y) = x is as fast as y
      * + EX 1 If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

We know A is n times faster than B if

Performance (A)/ Performance(B) = Execution Time(B)/ Execution Time(A) = n

15/10 = 1.5 times faster than B

* Measuring Performance
  + - CPU execution time for program = CPU clock cylces for a program/ clock rate
    - CPU execution time for program = CPU clock cylces for a program/ clock cycle time
    - CPU Clock cycles = instructions for a program x average clock cycles per instruction
    - CPU Time = (Instruction Count X CPI)/ Clock Rate

CHAPTER 2

* instruction set – vocab of commands understood by a given architecture.
* Stored program concept- idea that instructions and data of many types can be stored in memory as numbers leading to the stored program computer

2.2Operations of the Computer Hardware

* + MIPS assembly language is what the computer performs. Only three variables at a time.
    - EX – add a,b,c - add variables of b and c and put it into a
    - EX – add a,b,c # sum of b and c is placed in a
    - add a,a,d # Sum of b,c and d is now placed in a

Page 78

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CATEGORY | INSTRUCTIONS | EXAMPLE | MEANING | COMMANDS |
| Arithmetic | Add | Add $s1,$s2,$s3 | $s1=$s2+$s3 | Three register operands |
| Data Transfer | Load word | Lw $s1,20 ($s2) | $s1 = Memory[$s2+20] | Word from memory to register |
| Logical | And | And $s1,$s2,$s3 | $s1=$s2 & $s3 | Three reg operands, bit by bit AND |
| Conditional Branch | Branch on equal | Beq $s1,$s2,25 | if($s1==$s2) go to PC + 4+ 100 | Equal test , PC relative branch |
| Unconditional Jump | Jump | J 2500 | Go to 10000 | Jump to assigned address |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

* Book Example
  + f = (g+h) – (i+j);
  + Solution

add t0,g,h

add t1,i,j

sub f,t0,t1

* 1. Operands of the Computer Hardware
* Size of a register in the MIPS architecture is 32 bits
* word – the natural unit of access in a computer usually a group of 32 bits. Corresponds to the size of a register in the MIPS architecture
* Book Example
  + f = (g+h) – (i+j); variables f,g,h,i,j are assigned to $s0,$s2,$s3,$s4.
  + What is the compiled MIPS code?
  + Solution

add t0,$s1,$s2

add t1,$s3,$s4

sub $s0,$t0,$t1

* Memory Operands
  + data transfer instruction – a command that moves data between memory and registers
  + address – a value used to delineate the location of a specific data element within a memory array
* Book Example
  + A is an array of 100 words and the compiler has associated the variables g and h with the registers $s1 and $s2 as before. Lets assume that the starting address or base address of the array is in $s3. Compile this C assignment statement:
  + Solution

lw $t0,8($s3) #Temp reg $t0 gets A[8]

add $s1,$s2,$t0 # g=h+A[8]

* Hardware/ Software Interface
  + alignment restriction – requirement that data be aligned in memory on natural boundaries
* Book Example
  + Assume variable h is associated with register $s2 and the base address of the array A is in $s3. What is the MIPS assembly code for the C assignment statement. A[12] = h + A[8];
  + Solution

lw $t0,32($s3) # temporary reg $t0 gets A[8]

add $t0,$s2,$t0 # temporary reg $t0 gets h + A[8]

sw $t0,48($s3) # stores h + A[8] into A[12]

* Hardware/ Software Interface
  + Compiler tries to keep most frequently used variables in registers and places rest in memory
  + Spilling registers- put less common used variables into memory
  + MIPS arithmetic instruction reads two registers, operates, and then writes out the result
  + MIPS data transfer instruction reads one operand or writes one operand without operating on it
* Constant or Immediate Operands
  + EXAMPLE – add constant 4 to register$s3

lw $t0, AddrConstant4($s1) # $t0 = constant 4

add $s3,$s3, $t0 # $s3 = $s3 + $t0($t0 ==4)

OR

addi $s3,$s3, 4 # $s3 = $s3 + 4

* + Add immediate offers versions of the instructions in which operand is a constant.
  + $zero = value 0
  1. Signed and unsigned Numbers
* binary digit or binary bit- two numbers in base 2,0,1 that are the components of information
* EXAMPLE

1011

= (1 x 2^3) + (0 x 2^2) + (1x2^1) + (1x2^0)

= 8 + 0 + 2 + 1

= 11

* least significant bit – rightmost bit in a MIPS word 0
* most significant bit – leftmost bit in a MIPS word 31
* leading 0 means that the number is positive
* leading 1 means that the number is negative
* Book Example

What is the decimal value of this 32 bit twos complement number?

1111 1111 1111 1111 1111 1111 1111 1100

Solution

(1x-2^31) + (1x2^30) …. (0x2^0)

= -2,147,483,644 + 2,147,483,644

= - 4

OR 2 + 1 + 1 = -4

* Find the Negative of a Value Trick
  + Invert every 0 to 1 and add 1 to the result
* Book Example

Negate 2 = 0000 0000 0000 0000 0000 0000 0000 0010

1111 1111 1111 1111 1111 1111 1111 1101

+ 1

1111 1111 1111 1111 1111 1111 1111 1110

- 2 = 1111 1111 1111 1111 1111 1111 1111 1110

* 1. Representing Instructions in the Computer
* Registers $s0-$s7 map into registers 16-23
* Registers $t0-$t7 map into registers 8 -15
* Each segment of an instruction is called a field
* First and last field tells MIPS instructions
* Second field gives number of register that is first source operand of the additional operation
* Third field gives other source operand for additionalFourth field contains number of register that is to receive the sum
* 000000 = 6 bits 1000011 = 6 bits
* instruction format – form of representation of an instruction composed of field of binary numbers.
* Machine language – binary representation used for communication within a computer system
* hexadecimal- numbers in base 16

|  |  |
| --- | --- |
| HEXADECIMAL | BINARY |
| 0 hex | 0000 |
| 2 hex | 0010 |
| 3 hex | 0011 |
| 4 hex | 0100 |
| 5 hex | 0101 |
| 6 hex | 0110 |
| 7 hex | 0111 |
| 8 hex | 1000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Op | Rs | Rt | Rd | Shamt | Funct |
| 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

Op – basic operation of the instruction. Field that denotes the op and format of an instruction

rs – first register source operand

rt – second register source operand

shamt – shift amount

funct – function, this field often called function code. Selects specific variant of the operation in the op field.

R-type for registers or R-Format

I-type for immediate or I-Format

A 16 bit address means a load word instruction can load any word within a

region of 2^15 , 32,678 bytes

lw $t0,32($s3) # temporarily reg $t0 gets A[8]

# 19 for $s3 is placed in the rs field, 8 for $t0 is placed in the rt field and 32 is placed in the address field.

* Book Example

Translate MIPS Assembly language into Machine Language

If $t1 has the base of the array A and $s2 corresponds to h, the assignment statement

A[300] = h + A[300] is compiled into

lw $t0,1200($t1) # temp reg $t0 gets A[300]

add $t0,$s2,$t0 # temporary reg $t0 gets h + A[300]

sw $t0,1200($t1) # Stores h + A[300] back into A[300]

What is the MIPS machine language code for these instructions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OP | RS | RT | RD | ADDRESS/ shamt | FUNCT |
| 35 | 9 | 8 | 1200 | | |
| 0 | 18 | 8 | 8 | 0 | 32 |
| 43 | 9 | 8 | 1200 | | |

* Registers $s0-$s7 map into registers 16-23
* Registers $t0-$t7 map into registers 8 -15

lw = the op 35, $t1 = RS 9, $t0 = RT 8 destination register, 1200 = 300 x 4 or A[300] is the final field address

add = op 0, Three register operands (18,8 and 8) = $s2, $t0, $t0 rs, rt, and rd. The funct 32 is part of the add instructions

sw = op 43, $t1 = RS 9, $t0 = RT 8 destination register, 1200 = 300 x 4 or A[300] is the final field address

Page 100 is a good reference

* 1. Logical Operations

|  |  |  |
| --- | --- | --- |
| Logical Operations | Java Operations | MIPS Instruction |
| Shift left | << | Sll |
| Shift right | >> | Srl |
| Bit by bit AND | & | And, and I |
| Bit by bit OR | | | Or, ori |
| Bit by bit NOT | ~ | nor |

Example:

sll $t2,$s0,4 #reg $t2 = reg $s0 << 4 bits

* AND – two operands that calculates a 1 only if there is a 1 in both operands

Example:

if $t1 = 1101 1100 0000

$t2 = 1100 0000 0000

and $t0, $t1, $t2 # reg $t0 = reg $t1 & reg $t2

= 1100 0000 0000

* OR – two operands that calculates a 1 if there is a 1 in either operand Example:

if $t1 = 0000 1101 1100 0000

$t2 = 0011 1100 0000 0000 Example:

* if $t1 = 1101 1100 0000
* $t2 = 1100 0000 0000

or $t0,$t1,$t2 # reg $t0 = reg $t1 | reg $t2

= 0011 1101 1100 0000

* Not – two operands that inverts the bits. Every 1 becomes a 0 and vice versa
* Nor – calculates a 1 only if there is a 0 in both operands

page 105

# readstr.asm

# Registers used:

# a0 - syscall argument for target read address

# a1 - syscall argument for read length

# v0 - syscall parameter

# t0 - userd to hold length of user string

.text

main: # Execution starts at main

la $a0, usrstr # Load the destination address into a0

li $a1, 0x40 # Load the 64 byte limit into a1

li $v0, 8 #syscall parameter for read\_string

syscall # execute syscall

## Store the length of user string

move $t0, $v0 # The return value of read\_str9ing is the number of # characters read

## Echo the user string with a label from below

la $a0, restlbl # get response message address

li $v0, 4 # pring string syscall

syscall # execute syscall

la $a0, usrstr # get the users entered value

li $v0, 4 # pring string syscall

syscall # execute syscall

## Print the length of user string with a label from below

move $a0, $t0 # a0=t0

li $v0,1 # print int syscall

syscall

la $a0, lenlbl # Get the length message address

li $v0, 4 # Print string syscall

syscall

## Exit the program

li $v0, 10 # syscall code 10 is exit

syscall # Look in $v0 and execute the syscall

ldata

usrstr: .space 0x40 # reserved 64 bytes of memory

lenlbl: .asciiz \* characters long.\m”

restlbl: .asciiz “ You entered: “

New programmers

# readstr.asm

# Registers used:

# a0 - syscall argument for target read address

# a1 - syscall argument for read length

# v0 - syscall parameter

# t1 - pointer to user string

# t2 - current character

# t0 - userd to hold length of user string

.text

main: # Execution starts at main

la $a0, usrstr # Load the destination address into a0

li $a1, 0x40 # Load the 64 byte limit into a1

li $v0, 8 #syscall parameter for read\_string

syscall # execute syscall

## Store the length of user string

lw $t0, $zero # t0=0

l la $t1, usrstr # Get a pointer to user string

starlen\_loop:

## look at each character if new line, end loop, else t0++

lb $t2, 0($t1) # Read what string pointer sees t2 = &t1

beq $12,10, pool\_melrts # If t2 = new line exit loop

addi $t0,1 # increment length

addi $t1,1 # Increment the string pointer

b strlen\_loop # loop back to beginning

pool\_melrts:

## Echo the user string with a label from below

la $a0, restlbl # get response message address

li $v0, 4 # pring string syscall

syscall # execute syscall

la $a0, usrstr # get the users entered value

li $v0, 4 # pring string syscall

syscall # execute syscall

## Echo the string length with a label from below

move $a0, $t0 # move string length to syscall arg

li $v0, 1 # pring int syscall

syscall # execute syscall

la $a0, lenlbl # get the address of length label

li $v0, 4 # pring string syscall

syscall # execute syscall

## Exit the program

li $v0, 10 # syscall code 10 is exit

syscall # Look in $v0 and execute the syscall

ldata

usrstr: .space 0x40 # reserved 64 bytes of memory

lenlbl: .asciiz \* characters long.\m”

restlbl: .asciiz “ You entered: “

.text

main: # execution starts at main

la $a0, hello # load the address of hello string to $a0

jal print\_string # call print string function

la $a0, hello # load the address of hello string to $a0

jal print\_string # call print string function

# Lets print some of our data segment as integers

la $t0, hello # load the address of the beginning of data seg

lb $a0, 1($t0) # load the value 1 after the beginning of data

li $v0,1 # Print int syscall

syscall #execution begins

lw $a0,hello #load the address of beginning of data

li $v0,1 #syscall print\_int

syscall # print the word

la $a0, nline #address of new line

jal print\_string #Call print\_string function

li $v0,10 #syscall exit

syscall #be out

print\_string:

li $v0,4 # print string syscall

syscall #execute

jr $ra

EXAMPLE IN CLASS -

move $a0, $v0 # $v0 keeps the input value

jal FACT # call fact(), ra<- next PC

la $a1, result #S result is stored in $v'

sw sv0, 0($a1) # store value into memory

8 and 4 > True

1000

0100

0000

1101

1111

1101

Random Number Generator for MIPS

ON Wikipedia page

unit get\_random()

{

6)

}

MW | MZ | G1 | G2| Rand |

71 23 7 8 X

14 26 60 + 14 = 74

29 50 0 + 29 = 29

65 5 50 + 65 = 115

2 + 3(8) = 26

7 + 1(7) = 14

2 + 6(8) = 50

1 + 4(7) = 29

Rock paper scissors coding

gt

Class Notes 3.20.12

Read Chapter 2