Computer Organization and Design

Assembly & Simulation

Henry Fuchs

Slides adapted from Montek Singh, who adapted them
from Leonard McMillan and from Gary Bishop
Back to McMillan & Chris Terman, MIT 6.004 1999

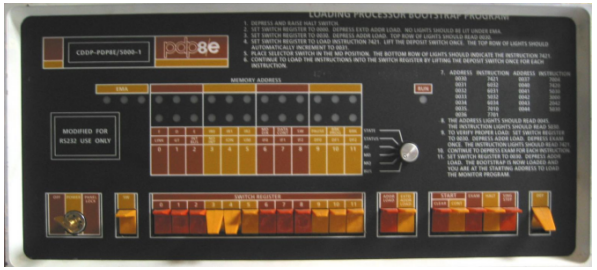
Thursday, Feb. 19, 2015
Lecture 7

Today

- Assembly programming
  - structure of an assembly program
  - assembler directives
  - data and text segments
  - allocating space for data
- MIPS assembler: MARS
  - development environment
- A few coding examples
  - self-study

What is an Assembler?

- A program for writing programs
- Machine Language:
  - 1's and 0's loaded into memory. (Did anybody ever really do that?)
- Assembly Language:



Front panel of a classic PDP8e. The toggle switches were used to enter machine language.

```
    .globl main
main:
    subu $sp, $sp, 24
    sw   $ra, 16($sp)
    li   $a0, 18
    li   $a1, 12
    li   $a2, 6
    jal  tak
    move $a0, $v0
```

Symbolic SOURCE text file

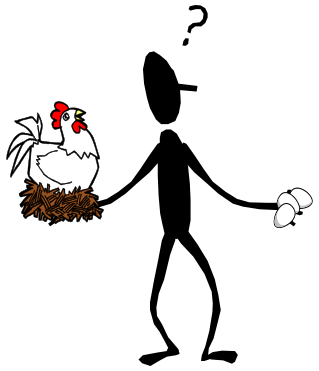
→ ASM →

ASSEMBLER Translator program

```
01101101
11000110
00101111
10110001
.....
```

Binary Machine Language

STREAM of bits to be loaded into memory



Assembler:

1. A Symbolic LANGUAGE for representing strings of bits
2. A PROGRAM for translating Assembly Source to binary

Assembly Source Language

An Assembly SOURCE FILE contains, in symbolic text, values of successive bytes to be loaded into memory... e.g.

`.data 0x10000000`	Specifies "current" address, i.e., start of data
`.byte 1, 2, 3, 4`	Four byte values
`.byte 5, 6, 7, 8`	Another four byte values
`.word 1, 2, 3, 4`	Four word values (each is 4 bytes)
`.asciiz "Comp 411"`	A zero-terminated ASCII string
`.align 2`	Align to next multiple of 2^2
`.word 0xfeedbeef`	A hex-encoded word value

Resulting memory dump:

```
[0x10000000]    0x04030201  0x08070605  0x00000001  0x00000002
[0x10000010]    0x00000003  0x00000004  0x706d6f43  0x31313420
[0x10000020]    0x00000000  0xfeedbeef  0x00000000  0x00000000
```

Notice the byte ordering. This MIPS is "little-endian" (The least significant byte of a word or half-word has the lowest address)

Assembler Syntax

- Assembler DIRECTIVES = Keywords prefixed with '.'
  - Control the placement and interpretation of bytes in memory

.data <addr>	Subsequent items are considered data
.text <addr>	Subsequent items are considered instructions
.align N	Skip to next address multiple of 2^N

  - Allocate Storage

.byte $b_1, b_2, \ldots, b_n$	Store a sequence of bytes (8-bits)
.half $h_1, h_2, \ldots, h_n$	Store a sequence of half-words (16-bits)
.word $w_1, w_2, \ldots, w_n$	Store a sequence of words (32-bits)
.ascii "string"	Stores a sequence of ASCII encoded bytes
.asciiz "string"	Stores a zero-terminated string
.space n	Allocates n successive bytes

  - Define scope

.globl sym	Declares symbol to be visible to other files
.extern sym size	Sets size of symbol defined in another file (Also makes it directly addressable)

More Assembler Syntax

- Assembler COMMENTS
  - All text following a '#' (sharp) to the end of the line is ignored
- Assembler LABELS
  - Labels are symbols that represent memory addresses
    - labels take on the values of the address where they are declared
    - labels can be for data as well as for instructions
  - Syntax: <start_of_line><label><colon>

```
.data 0x80000000
i:      .word 1                 # a data word

.text 0x00010000
start:  add     $3, $4, $2      # an instruction label
        sll     $3, $3, 8
        andi    $3, $3, 0xff
        beq     ..., ..., start
```

Even More Assembler Syntax

- Assembler PREDEFINED SYMBOLS
  - Register names and aliases

    `$0-$31, $zero, $v0-$v1, $a0-$a3, $t0-$t9, $s0-$s7, $at, $k0-$k1, $gp, $sp, $fp, $ra`
- Assembler MNEMONICS
  - Symbolic representations of individual instructions

    `add, addu, addi, addiu, sub, subu, and, andi, or, ori, xor, xori, nor, lui, sll, sllv, sra, srav, srl, srlv, div, divu, mult, multu, mfhi, mflo, mthi, mtlo, slt, sltu, slti, sltiu, beq, bgez, bgezal, bgtz, blez, bltzal, bltz, bne, j, jal, jalr, jr, lb, lbu, lh, lhu, lw, lwl, lwr, sb, sh, sw, swl, swr, rfe`
    - not all implemented in all MIPS versions
  - *Pseudo-instructions* (mnemonics that are not instructions)
    - `abs, mul, mulo, mulou, neg, negu, not, rem, remu, rol, ror, li, seq, sge, sgeu, sgt, sgtu, sle, sleu, sne, b, beqz, bge, bgeu, bgt, bgtu, ble, bleu, blt, bltu, bnez, la, ld, ulh, ulhu, ulw, sd, ush, usw, move, syscall, break, nop`
    - not real MIPS instructions; broken down by assembler into real ones

A Simple Programming Task

- Add the numbers 0 to 4 …
  - 10 = 0 + 1 + 2 + 3 + 4
- Program in "C":

```
int i, sum;

main() {
     sum = 0;
     for (i=0; i<5; i++)
          sum = sum + i;
}
```

- Now let's code it in ASSEMBLY

First Step: Variable Allocation

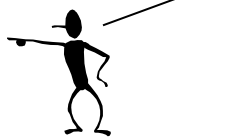
- Two integer variables (by default 32 bits in MIPS)

```
.data 0x80000000
.globl sum
.globl i
sum:     .space 4
i:       .space 4
```

- Thing to note:
  - ".data" assembler directive places the following words into the data segment
  - ".globl" directives make the "sum" and "i" variables visible to all other assembly modules (in other files)
    - could skip in single-file assembly programs
  - ".space" directives allocate 4 bytes for each variable
    - in contrast to ".word", ".space" does not initialize the variables

Actual "Code"

- Next we write ASSEMBLY code using instr mnemonics



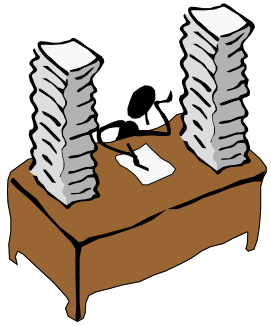
A common convention, which originated with the 'C' programming language, is for the entry point (starting location) of a program to named "main".

```
.text 0x10000000
.globl main
main:
      add   $8,$0,$0      # sum = 0
      add   $9,$0,$0      # for (i = 0; ...
loop:
      addu  $8,$8,$9      # sum = sum + i;
      addi  $9,$9,1       # for (...; ...; i++
      slti  $10,$9,5      # for (...; i<5;
      bne   $10,$0,loop
end:   ...
```

Bookkeeping:

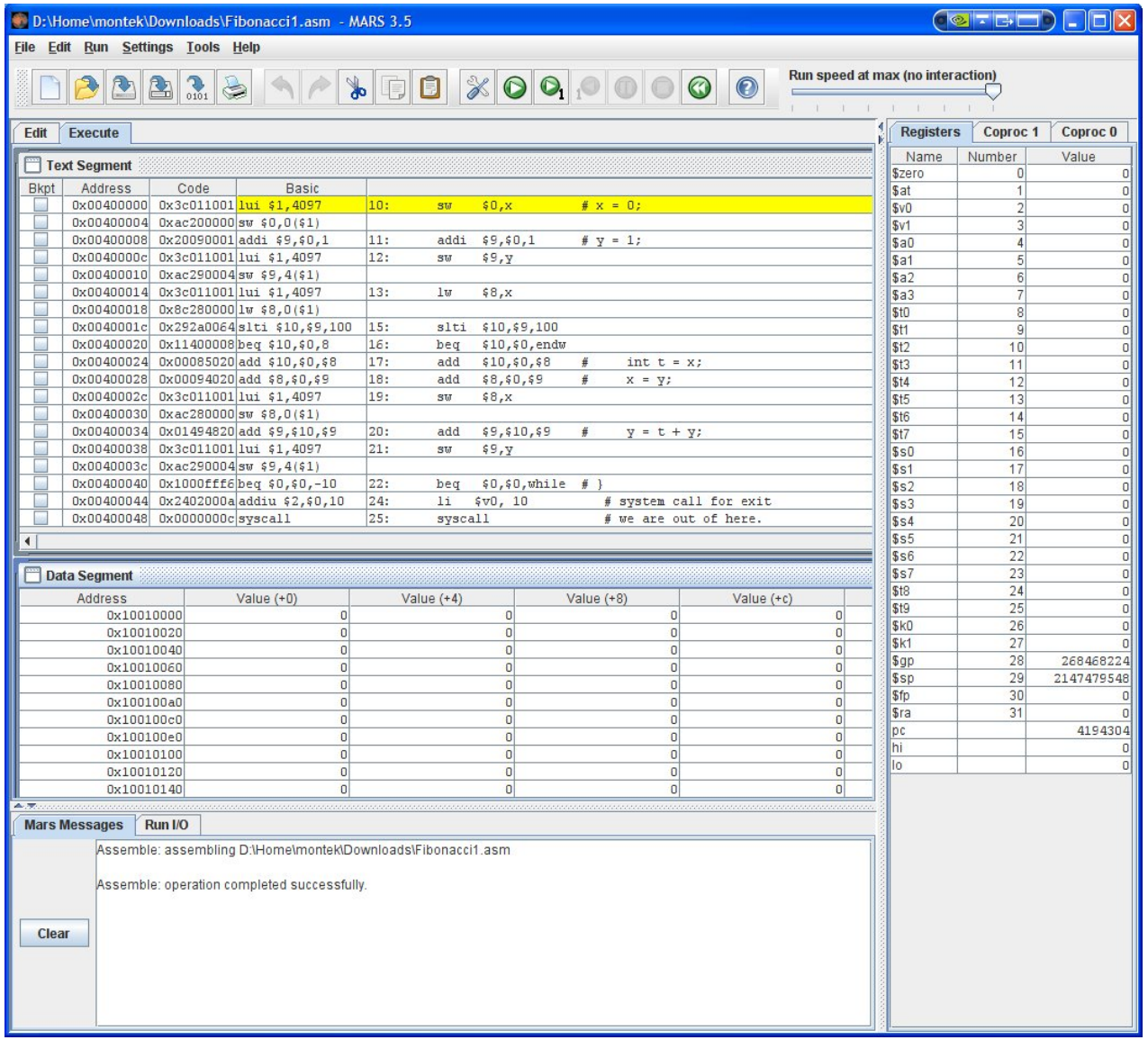
1) Register $8 is allocated as the "sum" variable
2) Register $9 is allocated as the "i" variable

We will talk about how to **exit** a program later



MARS

- MIPS Assembler and Runtime Simulator (MARS)
  - Java application
  - Runs on all platforms
  - Links on class website
  - Download it now!



A Slightly More Challenging Program

- Add 5 numbers from a list …
  - sum = $n_0 + n_1 + n_2 + n_3 + n_4$
- In "C":

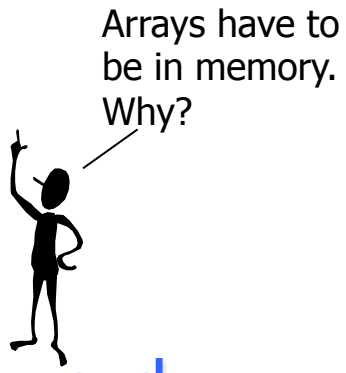
```
int i, sum;
int a[5] = {7,8,9,10,8};

main() {
     sum = 0;
     for (i=0; i<5; i++)
          sum = sum + a[i];
}
```

- Once more… let's code it in assembly

Variable Allocation

- Variables will be allocated to memory locations...
  - ... rather than registers
- This time we add the contents of an array

```
.data 0x0
sum:    .space 4
i:      .space 4
a:      .word 7,8,9,10,8
```



- Note: ".word" also works for an array of words
  - allows us to initialize a list of sequential words in memory
  - label represents the address of the first word in the list, or the name of the array

The New Code

- Note the small changes:

```
.data 0x0
sum:    .space 4
i:      .space 4
a:      .word 7,8,9,10,8
```

```
.text 0x3000
.globl main
main:
        sw    $0,sum        # sum = 0;
        sw    $0,i          # for (i = 0;
        lw    $9,i          # allocate register for i
        lw    $8,sum        # allocate register for sum
loop:
        sll   $10,$9,2      # covert "i" to word offset
        lw    $10,a($10)    # load a[i]
        addu  $8,$8,$10     # sum = sum + a[i];
        sw    $8,sum        # update variable in memory
        addi  $9,$9,1       # for (...; ...; i++
        sw    $9,i          # update memory
        slti  $10,$9,5      # for (...; i<5;
        bne   $10,$0,loop
end:    ...                 # code for exit here
```

Two different uses of register 10

A Little "Weirdness"

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,4097	12: sw $0,sum # sum = 0;
	0x00400004	0xac200000	sw $0,0($1)	
	0x00400008	0x3c011001	lui $1,4097	13: sw $0,i # for (i = 0;
	0x0040000c	0xac200004	sw $0,4($1)	
	0x00400010	0x3c011001	lui $1,4097	14: lw $9,i # allocate regis
	0x00400014	0x8c290004	lw $9,4($1)	
	0x00400018	0x3c011001	lui $1,4097	15: lw $8,sum # allocate regis
	0x0040001c	0x8c280000	lw $8,0($1)	
	0x00400020	0x00095080	sll $10,$9,2	17: sll $10,$9,2 # covert "i" to
	0x00400024	0x3c011001	lui $1,4097	18: lw $10,a($10) # load a[i]
	0x00400028	0x002a0821	addu $1,$1,$10	
	0x0040002c	0x8c2a0008	lw $10,8($1)	
	0x00400030	0x010a4021	addu $8,$8,$10	19: addu $8,$8,$10 # sum = sum + a[
	0x00400034	0x3c011001	lui $1,4097	20: sw $8,sum # update variabl
	0x00400038	0xac280000	sw $8,0($1)	
	0x0040003c	0x21290001	addi $9,$9,1	21: addi $9,$9,1 # for (...; ...;
	0x00400040	0x3c011001	lui $1,4097	22: sw $9,i # update memory
	0x00400044	0xac290004	sw $9,4($1)	

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c
0x10010000	0	0	7	
0x10010020	0	0	0	

Na...
$ze...
$at
$v0
$v1
$a0
$a1
$a2
$a3
$t0
$t1
$t2
$t3
$t4
$t5
$t6
$t7
$s0
$s1
$s2
$s3
$s4
$s5
$s6
$s7
$t8
$t9
$k0

The Assembler rewrote one of our instructions. What's going on?

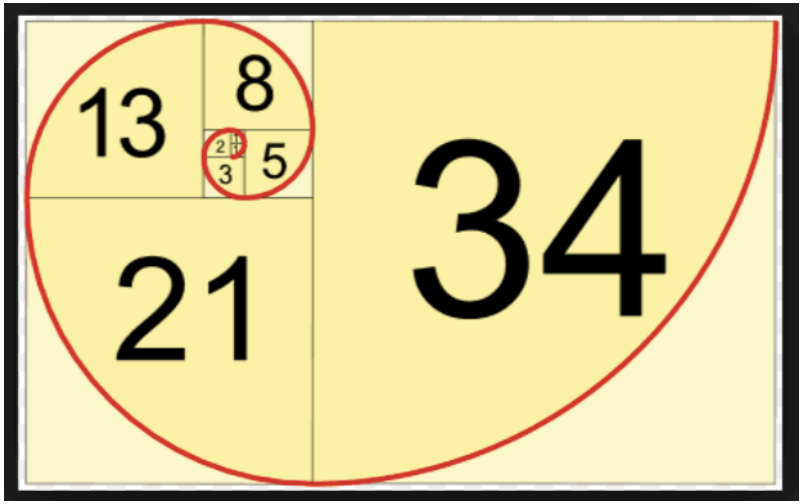
A Coding Challenge

- What is the largest Fibonacci number less than 100?
  - Fibonacci numbers:

    $F_{i+1} = F_i + F_{i-1}$

    $F_0 = 0$

    $F_1 = 1$



http://www.mathsisfun.com/numbers/fibonacci-sequence.html

  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …
- In "C":

```
int x, y;

main() {
     x = 0;
     y = 1;
     while (y < 100) {
          int t = x;
          x = y;
          y = t + y;
     }
}
```

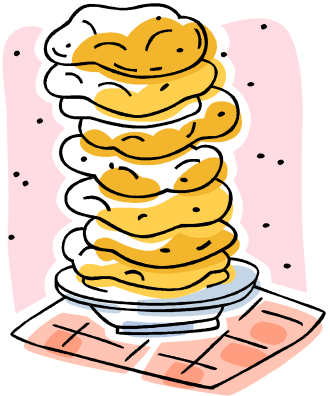
MIPS Assembly Code

* In assembly

```
.data 0x0
x:    .space 4
y:    .space 4

.text 0x3000
.globl main
main:
     sw    $0,x          # x = 0;
     addi  $9,$0,1       # y = 1;
     sw    $9,y
     lw    $8,x
while:                   # while (y < 100) {
     slti  $10,$9,100
     beq   $10,$0,endw
     add   $10,$0,$8     #     int t = x;
     add   $8,$0,$9      #     x = y;
     sw    $8,x
     add   $9,$10,$9     #     y = t + y;
     sw    $9,y
     j     while         # }
endw:
                         # code for exit here
                         # answer is in x
```

Next Class

- Parameterized Programs
- Procedures
- Stacks



- MIPS procedure linkage conventions