CSUN | CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

COMP 122

DR JEFF SOFTWARE
INDIE APP DEVELOPER

Rev 9-14-24

ASSEMBLY Programming

Labs 1-5

Dr Jeff Drobman

website → drjeffsoftware.com/classroom.html

email → jeffrey.drobman@csun.edu

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Lab Programs

Vol. 1

1. "Hello World": I/O (console and GUI) in MIPS & ARM
2. "Hello World" extended: loops, macros, functions/subroutines
3. Number systems and radix conversion
4. Computation 1: Fibonacci (add, arrays)
5. Computation 2: Factorials (mult, overflow)

Vol. 2

6. Floating-point: Temperature conversion
7. Prime numbers (algorithms)
8. Tic-tac-toe: Bit-wise operations (bit masks, shifts)

Vol. 3

9. Interrupt/Exception handler (trap with menu)
10. BCD on LED in real-time using MMIO, with commands
11. ISA design: create new *extended/pseudo* instructions

2	3	3	3	3	3	3	5	6	7	5	7
LAB	LAB	LAB	LAB	LAB	LAB	LAB	LAB	LAB	LAB	LAB	Proj
1	2	3	4	5	6	7	8	9	10	11	

Lab v. 1 Index

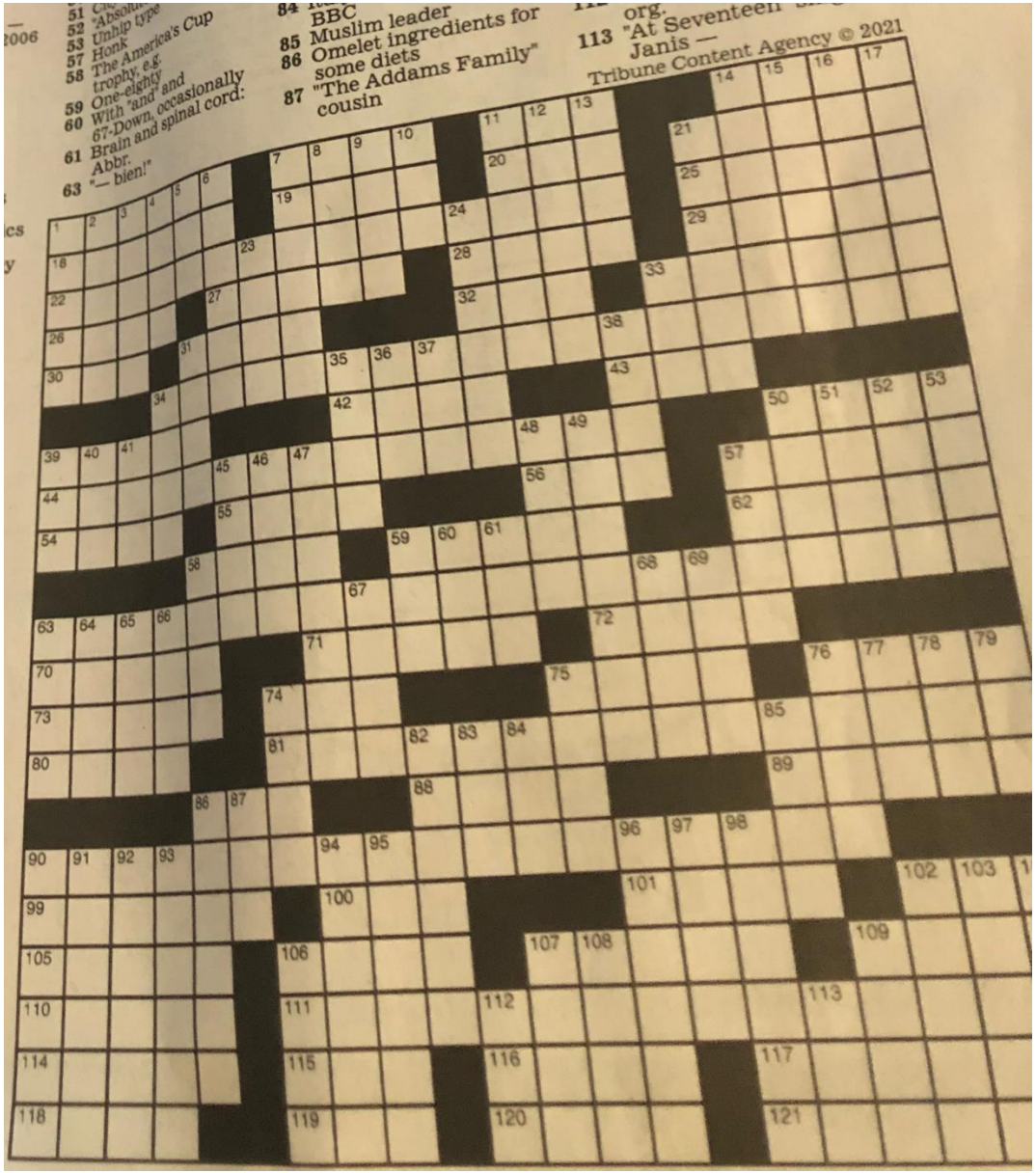
Lab

Assembly

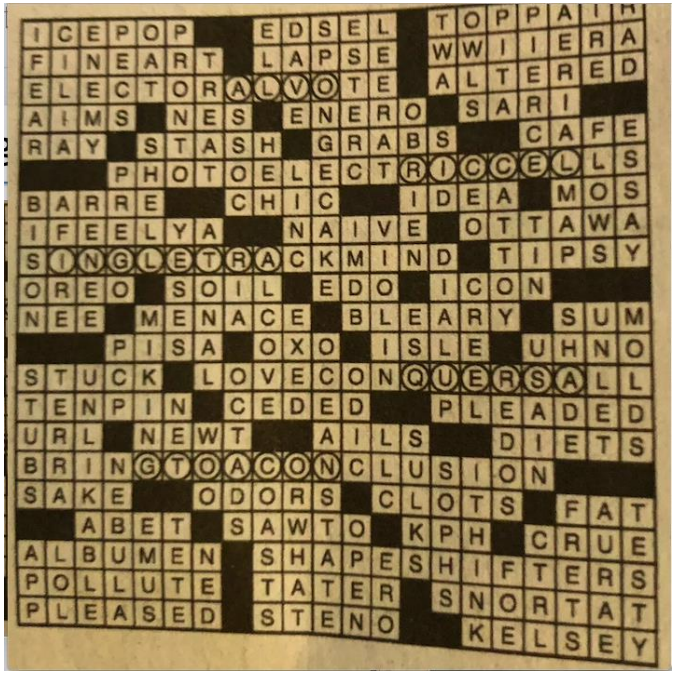
CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Programming as a Puzzle

DR JEFF SOFTWARE
INDIE APP DEVELOPER



Given Solution



CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Baseline Instruction Set

Rev Aug 2021

Computation

- ALU
  - ADD
  - SUB
  - AND
  - OR
  - XOR
  - NOT
- MULT/DIV [opt]
- BIT
  - SET/CLR
  - TEST
- COMPARE
  - CMP
- SHIFT
  - SHIFT (A, L)
  - ROTATE

Memory

- Reg-Reg
  - MOV
- Reg-Mem
  - LOAD
  - STORE
  - MOV

RISC

- Mem-Mem
  - MOV

CISC

- Stack
  - PUSH
  - POP

Program Control

- JUMP
  - JUMP/GOTO
- BRANCH
  - BRA
  - BRCC
  - LOOP
- CALL
  - CALL/CALR/JAL
  - RET/RETFIE
  - SYSCALL/SWI/TRAP
- NOP

System Control

- Reset
  - RESET
- Power
  - SLEEP/HALT

I/O

- I/O
  - IN
  - OUT

OLD

- Mem Mapped
  - MOV PORT
  - LOAD/STORE

NEW

Data Types: Java/C → MIPS

.data *Static data segment*

```
//numeric
int final x = 3;  → .eqv  x, 3 //immutable
int x = 3;  →  x:  .word  3
short y = 5;  → y:  .half  5
byte b = 2;  →  x:  .byte  2
float f = 45.0;  →  f:  .float  45
double d = 55.0;  →  d:  .double  55
//non-numeric
char ch = ‘a’; →  ch:  .ascii  “a”
String s = “hello” →  s:  .asciiz  “hello”
```

CPU Registers

- MIPS
- ARM

16-32 **GR**'s

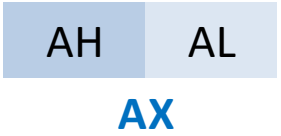
- Includes specials
  - Stack: SP, FP
  - Globals: GP
  - Zero
- 64-bit
  - *Paired* 32-bit

➢ **PC** dedicated

- x86
  - Intel
  - AMD

Dedicated Registers

- 4 Accumulators
  - A, B, C, D
- 4 Pointers
  - SI, DI
  - BP, SP
- 5 Memory Segments
  - CS, DS, ES, FS, GS, SS
- 64-bit
  - *Telescoping*



MARS

MARS (MIPS Assembler and Runtime Simulator)

Registers

Mars

MARS 4.5

File Edit Run Settings Tools Help

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	0x00000000
$12 (status)	12	0x0000ff11
$13 (cause)	13	0x00000000
$14 (epc)	14	0x00000000

Registers | Copro

Name	Float
$f0	0x00000000
$f1	0x00000000
$f2	0x00000000
$f3	0x00000000
$f4	0x00000000
$f5	0x00000000
$f6	0x00000000
$f7	0x00000000
$f8	0x00000000
$f9	0x00000000
$f10	0x00000000

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

GP Registers

Register use convention: Hennessy & Patterson

Figure 2.8.1: What is and what is not preserved across a procedure call (COD Figure 2.11).

If the software relies on the frame pointer register or on the global pointer register, discussed in the following subs preserved.

Preserved	Not preserved
Saved registers: `$s0–$s7`	Temporary registers: `$t0–$t9`
Stack pointer register: `$sp`	Argument registers: `$a0–$a3`
Return address register: `$ra`	Return value registers: `$v0–$v1`
Stack above the stack pointer	Stack below the stack pointer

- $a(0:3) *args*
- $at, $k(0:1) *reserved*
- $v(0:1) *values*
- $t(0-9) *temp*
- $s(0:7) *saved*
- $gp *global ptr*
- $sp *stack ptr*
- $fp *frame ptr*
- $ra *return addr*

Assembly

arm Developer IP PRODUCTS TOOLS AND SOFTWARE ARCHITECTURES INTERNET OF THINGS COMMUNITY SUPPORT DOCUMENTATION DOWNLOADS

Running Software on Your Target

Transfer your executable image to a target device.

Learn more

Raspberry Pi 4

Writing Arm Assembly Code

Learn Arm assembly language with our curated resources.

Learn more

ARMsim

Developing Embedded Software

Building Your First Embedded Image

Retargeting Output to UART

Creating an Event-Driven Embedded Image

Event driven

Assembly

Learning about the instruction set

Assembly instructions are the fundamental building blocks of any program. If you are going to write assembly code, you will need to understand what instructions are available to you.

The precise set of available instructions for a particular device is called the instruction set. The Arm architecture supports three *Instruction Set Architectures* (ISAs): A64, A32 and T32. Use these resources for more information:

- Learn more about the different instruction sets supported by the Arm architecture.
- Use the ISA exploration tools to discover the available A64, A32, and T32 instructions in easy-to-browse XML and HTML formats.
- Refer to the Arm Architecture Reference Manuals (A-profile, R-profile, and M-profile) for definitive ISA details.

Assembly

Learning about assembly language

Although the instruction set reference materials described in the Overview are comprehensive, they do not provide the best starting point for beginners.

The following resources introduce the basic concepts of programming in Arm assembly language:

- The Cortex-A Series Programmer's Guide explains architectural fundamentals and an introduction to assembly language code, along with other useful information for programmers.
- Arm Assembly Language: Fundamentals and Techniques by William Hohl is a popular resource with the Arm University Program. This book discusses the basics of assembly language.
- Embedded Systems Fundamentals with Arm Cortex-M based Microcontrollers: A Practical Approach by Dr Alexander G. Dean includes a chapter correlating C programming features with those in assembly code.

The Arm Compiler 5 toolchain (executable name `armasm`) uses a different syntax for assembly code to Arm Compiler 6 (executable name `armclang`) and GNU (executable name `as`). Although the instructions are mostly the same regardless of toolchain, the syntax around the instructions varies.

Hello World in Assembly

Writing your first assembly program

Here are some examples that you can follow to get started with Arm assembly language.

- "Hello World" in Assembly is an Arm Community blog post that shows how to build a simple Arm assembly program with GCC, running either natively or with a cross-compiler.
- Assembling armasm and GNU syntax assembly code in the Arm Compiler Software Development Guide demonstrates another Arm assembly program, this time using the Arm Compiler 6 toolchain.
- Using the integrated assembler in the Arm Compiler User Guide provides another example.

Embedding Assembly in C

Mixing C, C++, and assembly code

Even though you can now write Arm assembly code, you probably do not want to use it to hand-code your entire application.

You will probably want to write a small number of key functions in assembly, and call those functions from your main application code. You might want to do this to make use of existing assembly code, but the rest of your project is in C or C++.

- Calling assembly functions from C and C++ in the Arm Compiler User Guide shows you how to make function calls from C/C++ code to assembly code using the Arm Compiler 6 toolchain.
- Beyond Hello World: Advanced Arm Compiler 5 Features provides a similar example using Arm Compiler 5 within the DS-5 IDE.

3-Level Example

C

```
    #include <stdio.h>
int
main (int argc, char *argv[])
{
    int i;
    int sum = 0;
    for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
    printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

Assembly--MIPS

```
addiu     $29, $29, -32
sw        $31, 20($29)
sw        $4, 32($29)
sw        $5, 36($29)
sw        $0, 24($29)
sw        $0, 28($29)
lw        $14, 28($29)
lw        $24, 24($29)
multu     $14, $14
addiu     $8, $14, 1
slti      $1, $8, 101
sw $8,    28($29)
mflo      $15
addu      $25, $24, $15
bne       $1, $0, -9
sw        $25, 24($29)
lui       $4, 4096
lw        $5, 24($29)
jal       1048812
addiu     $4, $4, 1072
lw        $31, 20($29)
addiu     $29, $29, 32
jr        $31
move      $2, $0
```

Figure 7.1.2: MIPS machine language code for a routine to compute and print the sum of the squares of integers between 0 and 100 (COD Figure A.1.2).

Machine -- binary

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

Assembly Example

PIC18F

```
MPASM  5.37                     LAB4.ASM   10-12-2015  21:21:12         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 #include <p18F458.inc>
                     00001         LIST
                     00002 ; P18F458.INC  Standard Header File, Version 1.10   Microchip T
Message[301]: MESSAGE: (Processor-header file mismatch.  Verify selected processor.)
                     01714         LIST
                     00002 #include <lab4.inc>
  00000100           00001 START EQU 0X100;address 256
000000               00002         org 0
000000 EF84 F000     00003         GOTO _start
000008               00004         org 0x08
000008 0010          00005         RETFIE
000018               00006         org 0x018
000018 0010          00007         RETFIE
                     00008
000100               00009         org START
000100 0000          00010 ZERO DB 0
000102 0001          00011 ONE DB 1
000104 000A          00012 TEN DB D'10'
000106 00FF          00013 ALL1 DB 0xFF
000108 0000          00014 _start NOP
                     00015
                     00016
                     00003
00010A EF85 F000     00004 loop GOTO loop
                     00005         end
MPASM  5.37                       LAB4.ASM   10-12-2015  21:21:12         PAGE  2
```

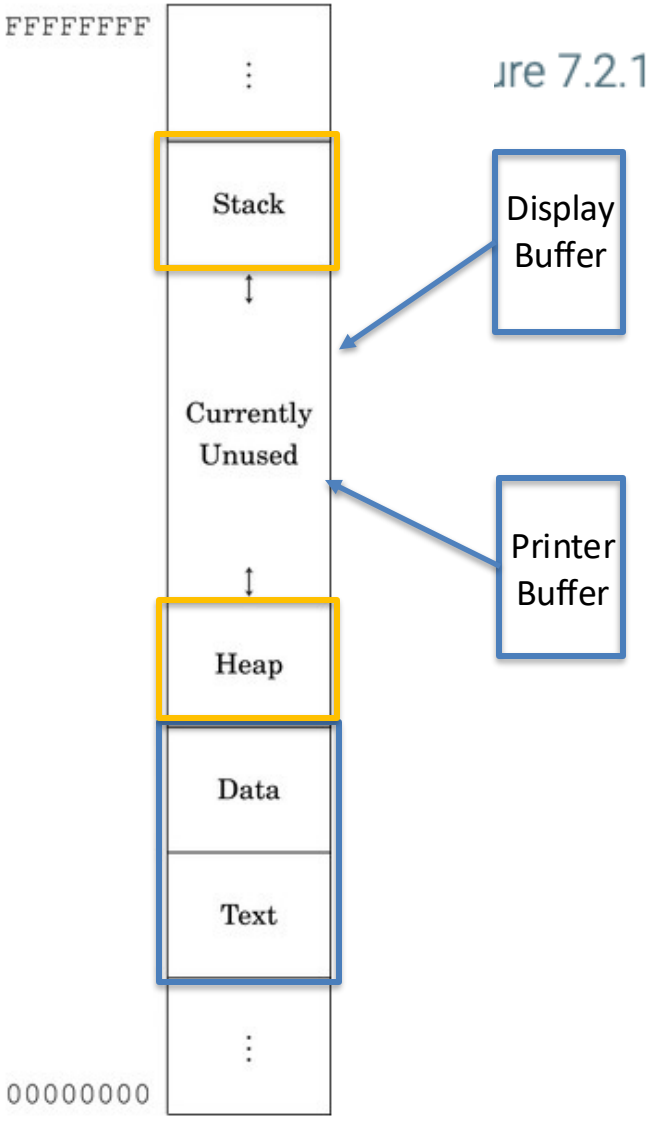
Space for code

```
        MOVLW   D'10'
        MOVWF   COUNTER
        LFSR    1,DATAPTR

LOOP    CLRF    POSTINC1
        DECF    COUNTER
        BNZ     LOOP

loop    GOTO    loop
        end
```

Memory Segments



ure 7.2.1: Object file (COD Figure A.2.1).

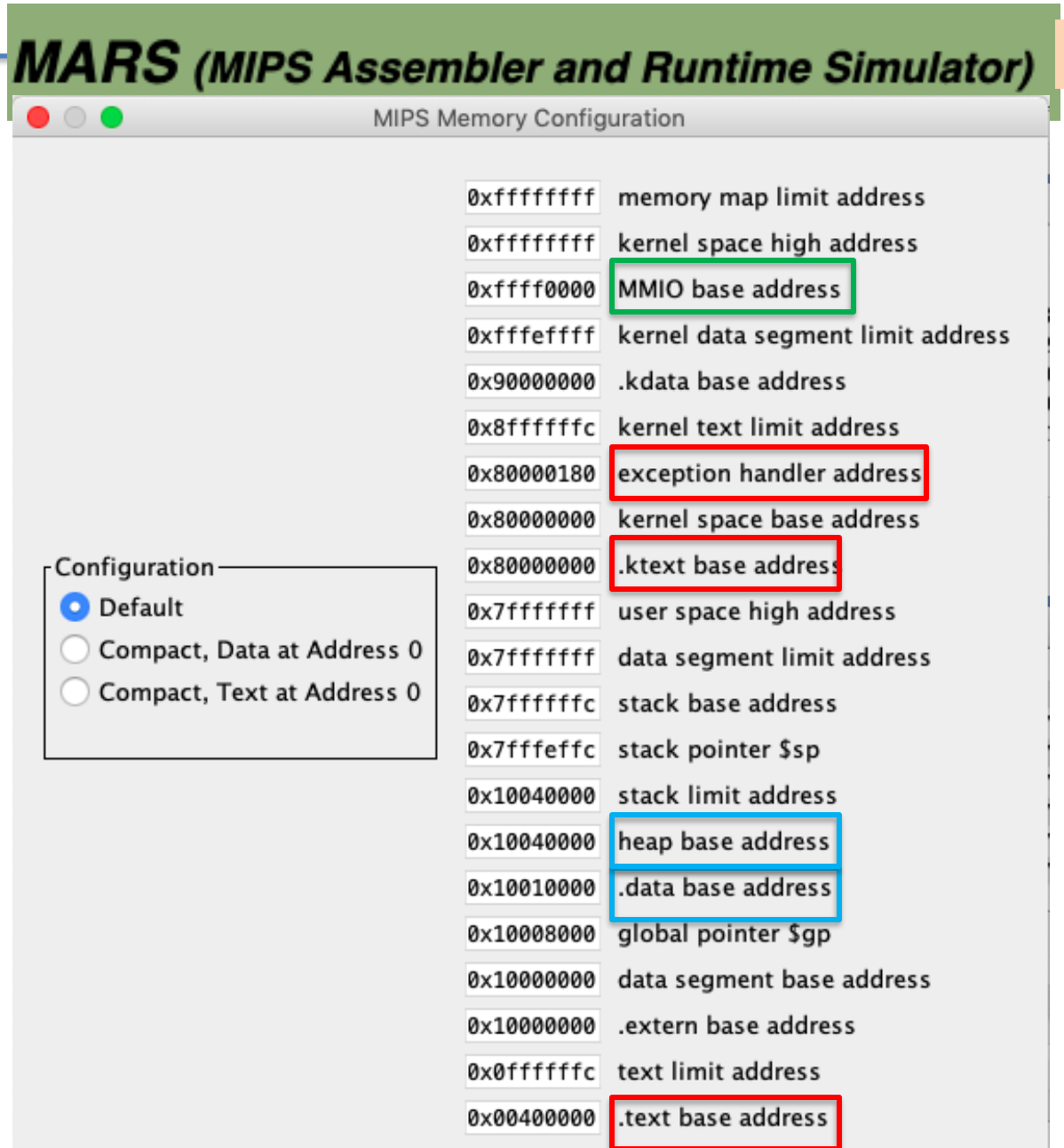
A UNIX assembler produces an object file with six distinct sections.

Object file header	Text segment	Data segment	Relocation information	Symbol table	Debugging information

Typical memory layout for a program with a 32-bit address space.

MARS

Memory Map



System Call (*syscall*)

MIPS

Figure 7.9.1: System services (COD Figure A.9.1).

SPIM

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $v0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

System Call (*syscall*)

SPIM

Services 1 through 17 are compatible with the SPIM simulator, other than Open File (13) as described in the Notes below the table. Services 30 and higher are exclusive to MARS.

50-59 are GUI boxes

MessageDialog	55	$a0 = address of null-terminated string that is the message to user $a1 = the type of message to be displayed: 0: error message, indicated by Error icon 1: information message, indicated by Information icon 2: warning message, indicated by Warning icon 3: question message, indicated by Question icon other: plain message (no icon displayed)

System Call (*syscall*)

SPIM provides a small set of operating system–like services through the system call
program loads the system call code (see the figure above) into register **$v0** and argu
point values). System calls that return values put their results in register **$v0** (or **$f0**
code prints **"the answer = 5"**:

```
    .data
str:
    .asciiz   "the answer = "
    .text

    li        $v0, 4          # system call code for print_str
    la        $a0, str        # address of string to print
    syscall                   # print the string

    li        $v0, 1          # system call code for print_int
    li        $a0, 5          # integer to print
    syscall                   # print it
```

Load address →

The **print_int** system call is passed an integer and prints it on the console. **print**

Macros with C *printf*

As an example, suppose that a programmer needs to print many numbers. The library routine **printf** accepts a format string and one or more values to print as its arguments. A programmer could print the integer in register $7 with the following instructions:

```
        .data
int_str: .asciiz "%d"
        .text
        la $a0, int_str # Load string address
                        # into first arg
        mov $a1, $7     # Load value into
                        # second arg
        jal printf      # Call the printf routine
```

Embedded format specifier for "printf"

The **.data** directive tells the assembler to store the string in the program's data segment, and the **.text** directive tells the assembler to store the instructions in its text segment.

However, printing many numbers in this fashion is tedious and produces a verbose program that is difficult to understand. An alternative is to introduce a macro, **print_int**, to print an integer:

```
        .data
int_str: .asciiz "%d"
        .text
        .macro print_int($arg)
        la $a0, int_str # Load string address into
                        # first arg
        mov $a1, $arg   # Load macro's parameter
                        # ($arg) into second arg
        jal printf      # Call the printf routine
        .end_macro
print_int($7)
```

.macro

Macros with C *printf*

The macro has a *formal parameter*, `$arg`, that names the argument to the macro. When the macro is expanded, the argument from a call is substituted for the formal parameter throughout the macro's body. Then the assembler replaces the call with the macro's newly expanded body. In the first call on `print_int`, the argument is `$7`, so the macro expands to the code

```
la   $a0, int_str
mov $a1, $7
jal printf
```

In a second call on `print_int`, say, `print_int($t0)`, the argument is `$t0`, so the macro expands to

```
la   $a0, int_str
mov $a1, $t0
jal printf
```

What does the call `print_int($a0)` expand to?

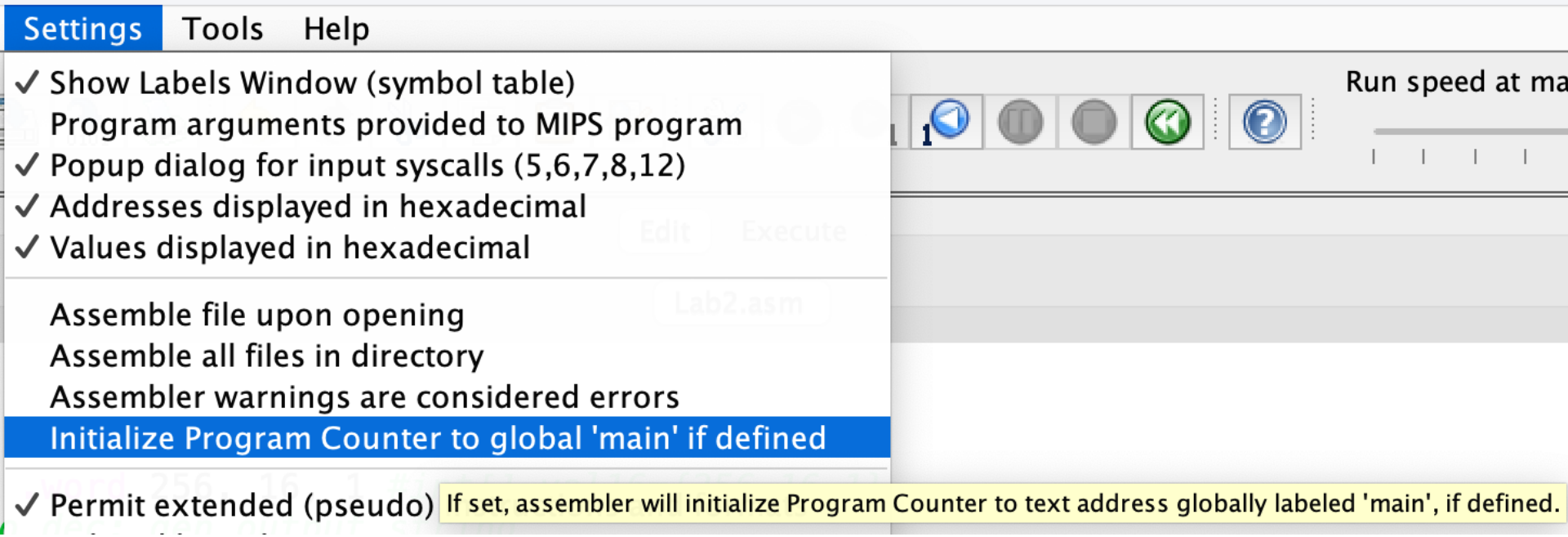
Answer

```
la   $a0, int_str
mov $a1, $a0
jal printf
```

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

.global main

Don't Use This!




```
.globl main
.text
main:
#convert hex to dec (use unrolled loop)
```

.globl Declare the listed label(s) as global to enable referencing from other files

Lab

Hello World

MIPS

Comparison: "Hello World"

Lab 1

C

```
#include <stdio.h>
void main (void) {
  printf("Hello world!\n");
}
```

C++

```
#include <iostream>
void main () {
  std::cout << "Hello world!\n";
}
```

Java

```
public class helloWorld {
public static void main (String[] args) {
  system.out.println ("Hello world!");
} }
```

Javascript

```
//myfile.js
Console.log("Hello world!");
```

Python

```
Print "Hello world!"
```

Comparison: "Hello World"

Basic

```
10 PRINT "Hello, world!"
20 END
```

note: line numbers!

VB

```
Public Sub Main()
  MsgBox "Hello, world!"
End Sub
```

OOP + **GUI**

C#

```
using System;

internal static class HelloWorld
{
    private static void Main()
    {
        Console.WriteLine("Hello, world!");
    }
}
```

OOP + console

DOS

```
@echo Hello World!
```

script (for console)

Comparison: "Hello World"

PHP

```php
<?php
print "Hello world!";
?>
```

- all console

x86 Assembly

```
.model small
.stack 100h

.data
msg     db      'Hello world!$'

.code
start:
        mov     ah, 09h
        lea     dx, msg
        int     21h
        mov     ax, 4C00h ;
        int     21h
end start
```

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Hello World in Java


```
1  /* CSUN COMP110 header
2  student: Jeff Drobman
3  ver date: 2-4-21
4  file:  Hello.java
5  Lab 1:  just Hello
6  */
7  //imports
8  import javax.swing.*;
9  //main class
10 public class Hello.java {
11 //static global DATA
12    static String hello = "Hello World!";
13 //main method
14    public static void main(String[] args) {
15       //OUTPUT
16       System.out.println(hello);//console
17       JOptionPane.showMessageDialog(null, hello);//GUI box
18    }//end main method
19 }//end class
```

Hello World in C

Standard C Implementation

A traditional introduction to many languages is the "Hello World" program. In C, this looks something like this:

```
#include <stdio.h>

int main(void) {
  printf("Hello, world.\n");
  return 0;
}
```

Hello World in C++

HelloWorld.cpp

```
//============================================================
// Name        : HelloWorld.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================

#include <iostream>
using namespace std;

int main() {
    cout << "!!!Hello World!!!" << endl; // prints !!!
    return 0;
}
```

JHD-HelloWo... | makefile | objects.mk | sources.mk

```
1  //============================================================
   Name        : JHD-HelloWorld.cpp
   Author      : JHD
4  // Version     :
5  // Copyright   : copyright JHD
6  // Description : Hello World in C++, Ansi-style
7  //============================================================
8
9  #include <iostream>
10 using namespace std;
11
12 int main() {
13     cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
14     return 0;
15 }
16
```

Hello World C++ Extended


```
#include <iostream>
using namespace std;

int main () {
    // Say HelloWorld five times
    for (int index = 0; index < 5; ++index)
      cout << "HelloWorld!" << endl;
    char input = 'i';
    cout << "To exit, press 'm' then the 'Enter' key." << endl;
    cin  >> input;
    while(input != 'm') {
        cout << "You just entered '" << input << "'. "
             << "You need to enter 'm' to exit." << endl;
        cin  >> input;
    }
    cout << "Thank you. Exiting." << endl;
    return 0;
}
```

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Hello World in Python


```
>>> print "Hello World!"
Hello World!
>>>
```

```
Last login: Fri Aug 16 13:38:24 on ttys000
Jeffreys-MacBook-Air:~ jhdphd$ python
Python 2.7.10 (default, Aug 17 2018, 17:41:52)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.0.42)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello World!"
Hello World!
>>>
```

Python

```
A. HISTORY OF THE SOFTWARE
==========================

Python was created in the early 1990s by Guido van Rossum at Stichting
Mathematisch Centrum (CWI, see http://www.cwi.nl) in the Netherlands
as a successor of a language called ABC.  Guido remains Python's
principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for
National Research Initiatives (CNRI, see http://www.cnri.reston.va.us)
in Reston, Virginia where he released several versions of the
software.

In May 2000, Guido and the Python core development team moved to
BeOpen.com to form the BeOpen PythonLabs team.  In October of the same
year, the PythonLabs team moved to Digital Creations (now Zope
Corporation, see http://www.zope.com).  In 2001, the Python Software
Foundation (PSF, see http://www.python.org/psf/) was formed, a
non-profit organization created specifically to own Python-related
Intellectual Property.  Zope Corporation is a sponsoring member of
the PSF.

All Python releases are Open Source (see http://www.opensource.org for
Hit Return for more, or q (and Return) to quit:
```

ASCII

Binary, hexadecimal, and decimal equivalents for each character in "Hello World"

Character	Binary	Hexadecimal	Decimal
H	01001000	48	72
e	01100101	65	101
l	01101100	6C	108
l	01101100	6C	108
o	01101111	6F	111
	00100000	20	32
W	01010111	57	87
o	01101111	6F	111
r	01110010	62	98
l	01101100	6C	108
d	01100100	64	100
NUL	00000000	00	0

Assembler *Endian*

est address. Some processors, such as the ARM, can be configured as either little-endian or big-endian. The Linux operating system, by default, configures the ARM processor to run in little-endian mode .

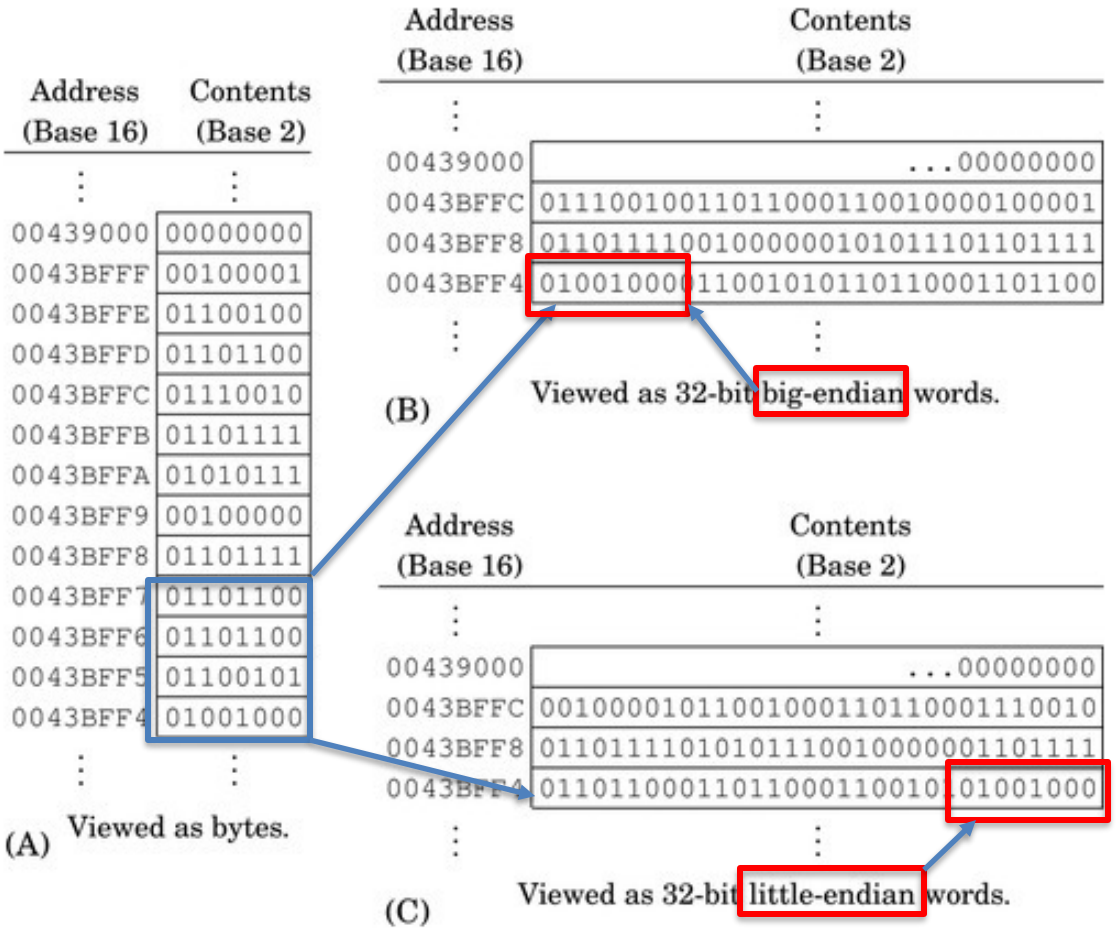


FIGURE 1.6 A section of memory.

Lab

LAB 1

Hello World

MIPS

Lab 1

Requirements

- Assemble into .data seg
  - *"Hello World\n"*
  - <your name>
- Print on Console & GUI
  - "Hello World!"
  - "Hello <your name>"

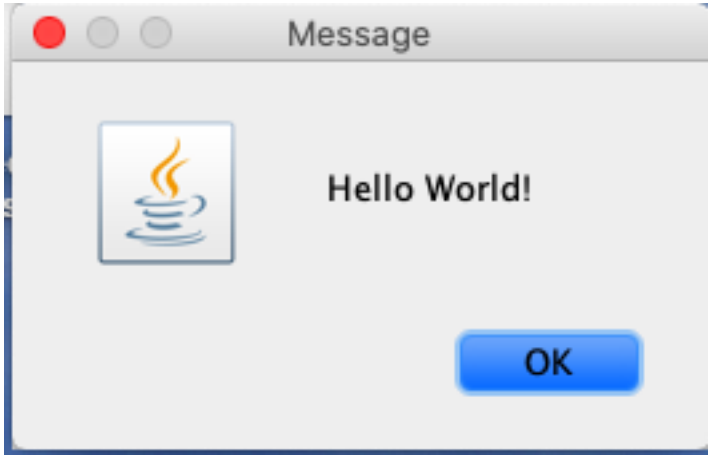
- End program
  - Use "done" macro (optional)

Java version of Lab 1

OUTPUT

```
 ----jGRASP exec: java Lab1Hello122
Hello World!

 ----jGRASP: operation complete.
```

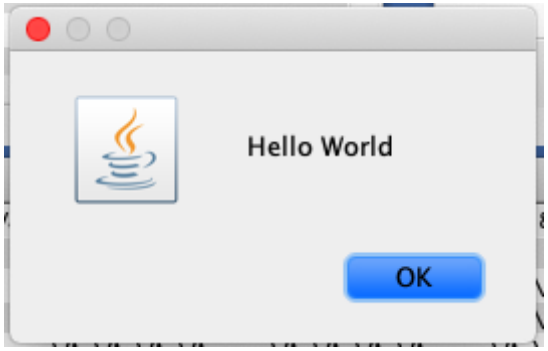


MIPS Assembly Output

Console prints

```
==*==
Hello World
Jeff D
Hello World
Jeff D

-- program is finished running --
```



Hello Jeff D

MIPS Macros

Termination

```
5   .macro done
6   li $v0, 10
7   syscall
8   .end_macro
```

define

```
34  nop # extra
35  done
36  break 0 # System.exit(0)
```

use

Don't use

- Considered an "error"

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

MIPS Assembly

MIPS Lab 1

mips-Hello.asm

```
1  ## Hello World
2  ## by Jeff Drobman
3  ##
4  .data
5  hello: .asciiz  "Hello World"
6  #
7  .text
8  lw $t1,hello
9  sw $t1,hello
10
```

Header

Static data

\n → newline

CODE Load-Store

Assembled Code

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x00001001	8: lw $t1,hello
	0x00400004	0x8c290000	lw $9,0x00000000($1)	
	0x00400008	0x3c011001	lui $1,0x00001001	9: sw $t1,hello
	0x0040000c	0xac290000	sw $9,0x00000000($1)	

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

MIPS Assembly

DR JEFF SOFTWARE
INDIE APP DEVELOPER

MIPS Lab 1

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x00001001	8: lw $t1,hello
	0x00400004	0x8c290000	lw $9,0x00000000($1)	
	0x00400008	0x3c011001	lui $1,0x00001001	9: sw $t1,hello
	0x0040000c	0xac290000	sw $9,0x00000000($1)	

Static data

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	l l e H	o W o	\0 d l r	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010020	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

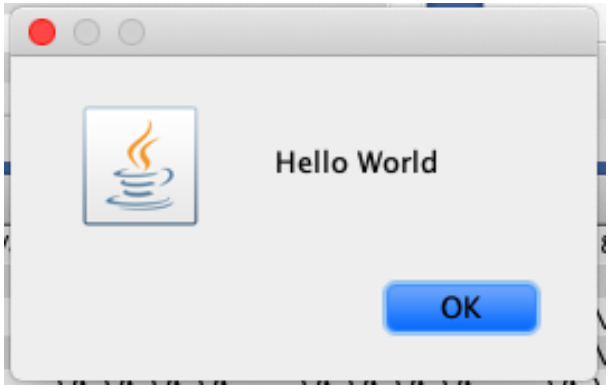
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

$sp	29	0x7fffeffc

MIPS Assembly: GUI Out

Use: Syscall **55**

MessageDialog	55	$a0 = address of null-terminated string that is the message to user $a1 = the type of message to be displayed: 0: error message, indicated by Error icon 1: information message, indicated by Information icon 2: warning message, indicated by Warning icon 3: question message, indicated by Question icon other: plain message (no icon displayed)




```
#output GUI msg
li $v0, 55 #GUI msg code
la $a0, hello
li $a1,1 #msg type is info
syscall
```

Memory Buffers: Output

55

```
46  #output GUI msg
47  li $v0, 55 #GUI msg code
48  la $a0, hello
49  li $a1,1 #msg type is info
50  syscall
```

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Va
0x10010000	* * = =	\0 \n = =	l l e H	o W o	\n d l r	
0x10010020	u p n I	t s t	g n i r	\0 \0 \0 :	\0 \0 \0 \0	
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x10010060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x10010080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x100100e0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	

0x10010000 (.data) Hexadecimal Addresses Hexa

Lab

LAB 1A ARM

Hello World

Port to

ARM

Dr Jeff Lab 1 ARM .data


```
@@ Lab 1 -- Hello World
@@ by Jeff Drobman
@@version: 1D >3:57 PM 11/14/2019
@-----------
@define
.equ exit, 0x11
.equ print, 2
@macros
.macro done
swi exit
.endm
@@---data---
.data
hello: .asciz  "Hello World\n"
.align 2 @align to word
name:  .asciz  "Jeff D\n"
```

Dr Jeff Lab 1 ARM Code


```
@--print on console using "@syscall=SWI"
ldr r0, =hello
SWI print
ldr r0, =name
SWI print
ldr r0, =heap @print entire string in heap
SWI print
@-----
@output GUI msg
mov v1, #55 @GUI msg code
ldr a1, =hello
mov a2, #1 @msg type is info
svc 55 @syscall
@------
nop @example
done @macro for exit
```

Hello World in Assembly

essors blog › "Hello World" in Assembly

ARM Gnu

```
    .syntax unified

    @ ------------------------------
.global main
main:
    @ Stack the return address (lr) in addition to a dummy register (ip) to
    @ keep the stack 8-byte aligned.
    push    {ip, lr}

    @ Load the argument and perform the call. This is like 'printf("...")' in C.
    ldr     r0, =message
    bl      printf

    @ Exit from 'main'. This is like 'return 0' in C.
    mov     r0, #0    @ Return 0.

    @ Pop the dummy ip to reverse our alignment fix, and pop the original lr
    @ value directly into pc — the Program Counter — to return.
    pop     {ip, pc}

    @ ------------------------------
    @ Data for the printf calls. The GNU assembler's ".asciz" directive
    @ automatically adds a NULL character termination.
message:
    .asciz "Hello, world.\n"
```

Call printf

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Hello World in Assembly

cessors blog › "Hello World" in Assembly

ARM Gnu

+ New

We can assemble and run this program using the following (on an Arm Linux-like platform):

```
gcc -o hello_world hello_world.s
$ ./hello_world
```

You should then see the text "Hello, world." on the console.

If you're using a cross-compiler (such as RVCT or the Code Sourcery edition of GCC) you'll need to run the first step on your PC — probably substituting `gcc` with something like `arm-none-linux-gnueabi-gcc` — and then copy the output binary to an Arm target before running the program itself.

Gnu Assembly: Hello World


```
1       .data
2  str:   .asciz  "Hello World\n" @ Define a null-terminated string
3
4       .text
5       .globl  main
6       /* This is the beginning of the main() function.
7          It will print "Hello World" and then return.
8       */
9  main:  stmfd   sp!,{lr}       @ push return address onto stack
10      ldr     r0, =str        @ load pointer to format string
11      bl      printf          @ printf("Hello World\n");
12      mov     r0, #0          @ move return code into r0
13      ldmfd   sp!,{lr}        @ pop return address from stack
14      mov     pc, lr          @ return from main
```

LISTING 2.1 "Hello World" program in ARM assembly

```
1  #include <stdio.h>
2  static char str[] = "Hello World\n";
3  int main()
4  {
5    printf(str);
6    return 0;
7  }
```

LISTING 2.2 "Hello World" program in C.

Hello World-Assembly *Listing*


```
ARM GAS  hello.S                 page 1

  Addr Code            Source
 1                     .data
 2 0000 48656C6C  str:  .asciz  "Hello World\n"
 2      6F20576F
 2      726C640A
 2      00
 3
 4                     .text
 5                     .globl  main
 6                     /* This is the beginning of the main() function.
 7                        It will print Hello World'' and then return.
 8                     */
 9 0000 00402DE9  main: stmfd   sp!,{lr} @ push return address onto stack
10 0004 0C009FE5       ldr     r0, =str @ load pointer to format string
11 0008 FEFFFFEB       bl      printf   @ printf("Hello World - %d\n",i);
12 000c 0000A0E3       mov     r0, #0   @ move return code into r0
13 0010 0040BDE8       ldmfd   sp!,{lr} @ pop return address from stack
14 0014 0EF0A0E1       mov     pc, lr   @ return from main
14      00000000

ARM GAS  hello.S                 page 2


DEFINED SYMBOLS
            hello.S:2      .data:0000000000000000 str
            hello.S:9      .text:0000000000000000 main
            hello.S:9      .text:0000000000000000 $a
            hello.S:14     .text:0000000000000018 $d

UNDEFINED SYMBOLS
printf
```

LISTING 2.3 "Hello World" assembly listing.

Assembler *Directives*


```
        .data
i:      .word   0
j:      .word   1
fmt:    .asciz  "Hello\n"
ch:     .byte   'A','B',0
ary:    .word   0,1,2,3,4
```

(A) Declarations in assembly

```
static int i = 0;
static int j = 1;
static char fmt[] = "Hello\n";
static char ch[] = {'A','B',0};
static int ary[] = {0,1,2,3,4};
```

(B) Declarations in C

FIGURE 2.1 Equivalent static variable declarations in assembly and C.

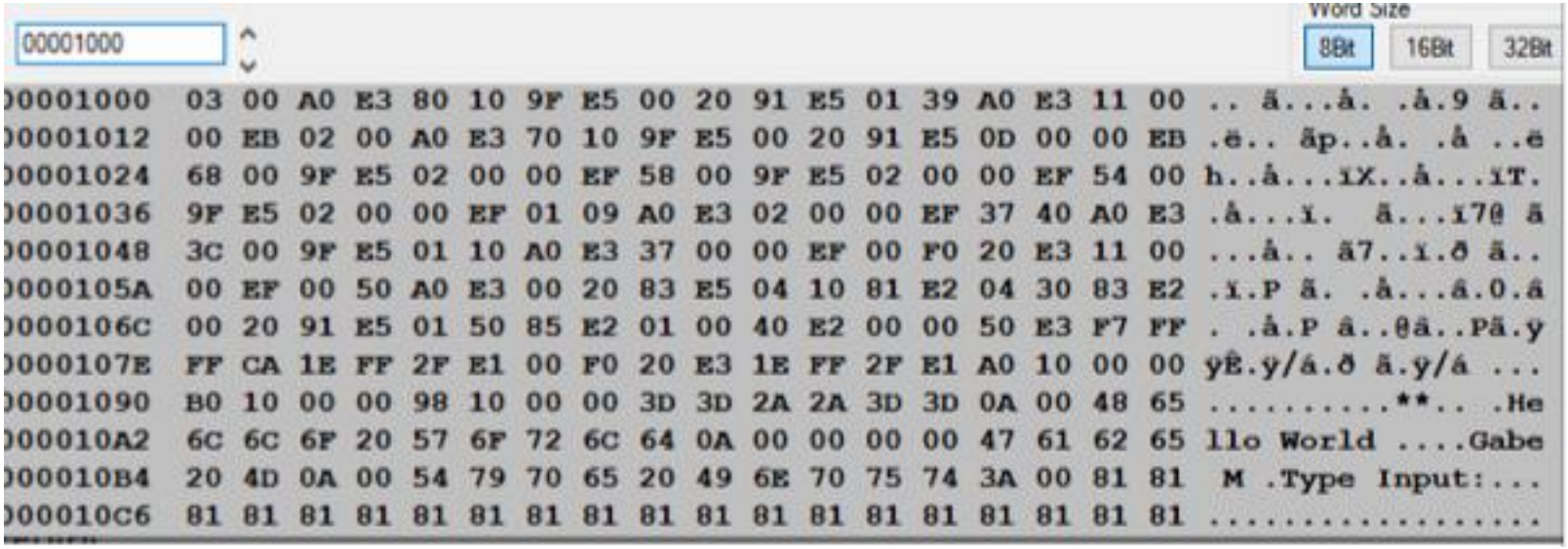
Lab 1A – ARMsim

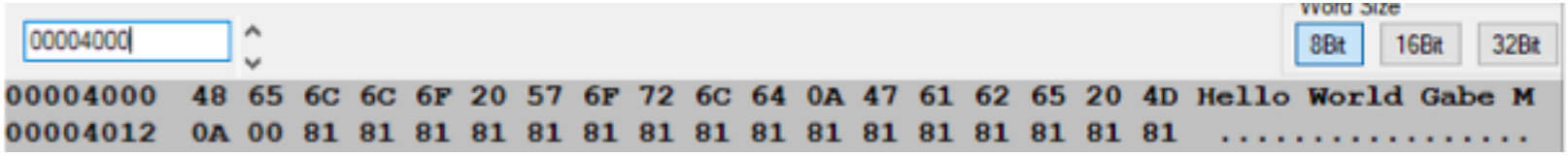

```
R0      :000010a0
R1      :00000001
R2      :65707954
R3      :00004014
R4      :00000037
R5      :00000002
R6      :00000000
R7      :00000000
R8      :00000000
R9      :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00011400
R14(lr):00001024
R15(pc):00001058
------------------
CPSR Register
Negative(N):0
Zero(Z)    :1
Carry(C)   :1
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
------------------
0x600000df
```

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Lab 1A – ARMsim


```
                        .data
00001098:2A2A3D3D       header: .asciz "==**==\n"
        :000A3D3D
                        .align 2 @align to word
000010A0:6C6C6548       hello: .asciz "Hello World\n"
        :6F57206F
        :0A646C72
        :00
```





Lab 1A – ARMsim

Lab 1D (ARM)

perfect/very good, and

1. Loop-- good use of "cmp" to set up branch.
2. GUI output: what happens when you execute SWI 55? You used reg "v1" but there is no such ARM reg.

```
@output GUI msg
mov v1, #55 @GUI msg code
ldr a1, =hello
mov a2, #1 @msg type is info
svc 55 @syscall
```

Svc 55

```
@subrountine:write String to heap($a0..$a3)
write_heap:
mov r5, #0 @counter
Loop: str a3, [a4] @count N= a1
add a2, a2, #4 @adjust ptrs
add a4, a4, #4
ldr a3, [a2] @load next word
add r5, #1
sub a1, #1 @count--
cmp a1, #0
Bgt Loop @EQ/NE/GE?
BX LR @return
@-----
printf: nop @stub
BX LR @return
.end
```

Cmp a1, #0

Lab 2

Requirements

- Add to Lab 1
  - GUI Input <your name>
    - Check input sub (optional)
  - Copy "Hello World!" (use *loop*) from *data* seg → into *heap* seg
    - Use "lw, sw" pairs
  - Print *heap* seg on console
  - Print both strings in GUI box (code 59)
  - Create & use "done" *macro*

Echo Name on Console


```
//INPUT
String name = JOptionPane.showInputDialog(prompt);//GUI box
//echo
System.out.println("Hello" + name);//console
```

```
----jGRASP exec: java Lab1Hello122
Hello World!
Hello Jeff
```

Setup Code


```
4   ##add:  sub, GUI IN: prompt+INbuf
5   #register map:
6   #$a1=string data, $a2=string ptr, $a3=heap ptr
7   #$a0=loop count (N)
8   .data
9   hello: .asciiz  "Hello World\n"
10  .align 2 #align to word
11  name:  .asciiz  "Jeff Drobman\n"
12  .align 2 #align to word
13  prompt:    .asciiz  "Input your name:"
14  .align 2 #align to word
15  ovfl: .asciiz "Error: max length exceeded!\n"
16  #define
17  .eqv heap, 0x10040000
18  .eqv mask3,0xff000000
19  .eqv hello_len, 3
20  .eqv name_len, 4
21  .eqv in_buf, 0x10040020 #input buffer
```

Java version of Lab 2

Header

CODE

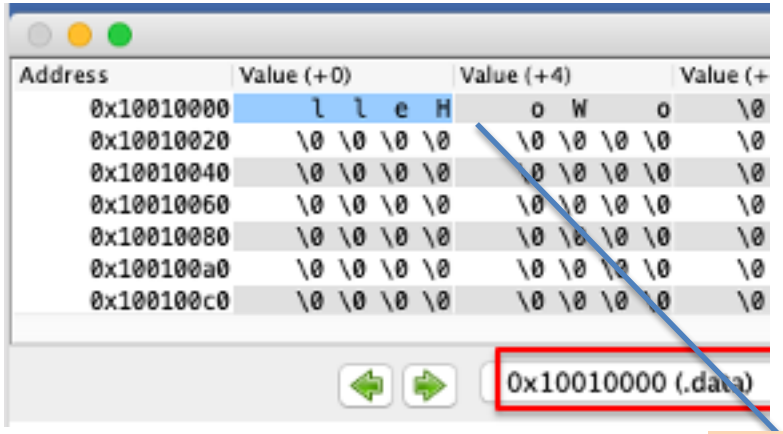
Static data

Copy to heap

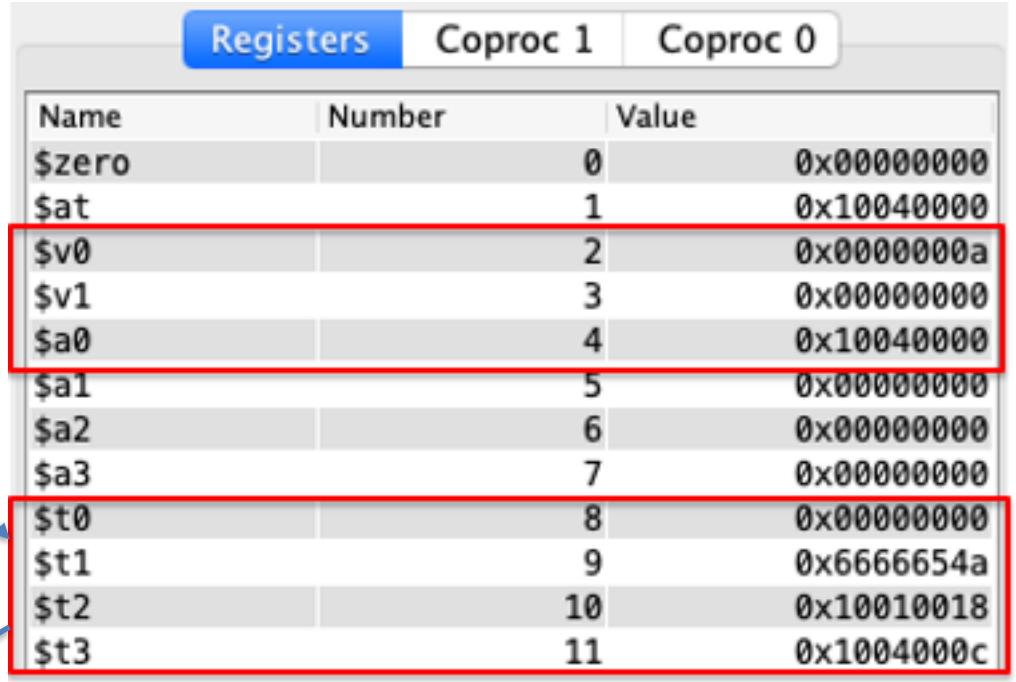
Output

```
1  /* CSUN COMP110 header
2  student: Jeff Drobman
3  ver date: 1-16-21
4  file:  Lab1Hello122.java
5  Lab 1A:  Hello basic
6  */
7  //imports
8  import java.util.*;
9  import javax.swing.*;
10 //main class
11 public class Lab1Hello122 {
12 //static global DATA
13    static String hello = "Hello World!";
14 //main method
15    public static void main(String[] args) {
16       //simulate a "heap" = dynamic DATA segment
17       char[] heap = new char[1000]; //reserve 1000 bytes
18       char[] charHello = hello.toCharArray();
19
20       //copy to heap (all words)
21       for(int i=0; i<hello.length(); i++)
22          heap[i] = charHello[i];
23
24       //OUTPUT
25       System.out.println(hello);//console
26       JOptionPane.showMessageDialog(null, hello);//GUI box
27
28    }//end main method
```

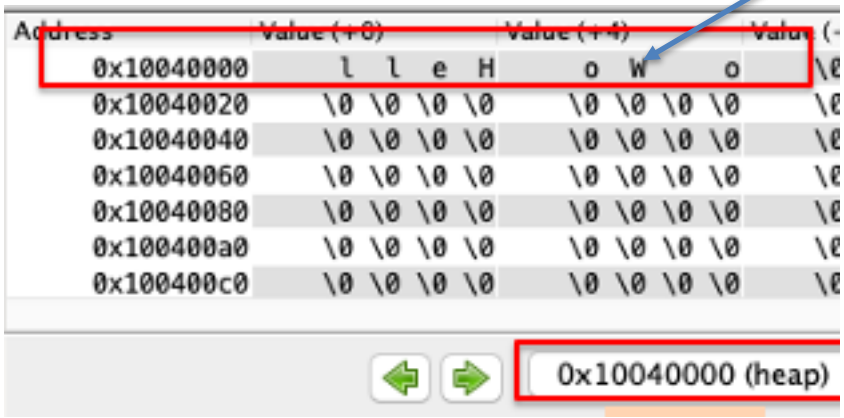
Memory "Move"



Load



Store



MIPS Assembly


```
1  ## Hello World
2  ## by Jeff Drobman
3  ##
4  #register map:
5  #$t1=string data, $t2=heap pointer
6  .eqv heapHi, 0x1004
7  .data
8  #heap: 0x10040000
9  hello: .asciiz  "Hello World\n"
10 .text
11 #store word 1 in heap
12 lw $t1, hello
13 lui $t2, heapHi
14 sw $t1, ($t2)
15 #store word 2 in heap
16 lw $t1, hello+4
17 add $t2, $t2, 4
18 sw $t1, ($t2)
19 #print on console using "Syscall 4"
20 li $v0, 4 #print code=
21 la $a0, hello #address (pointer)
22 syscall
23 nop #extra
24 break 0 #System.exit(0)
```

Setup (init static data)

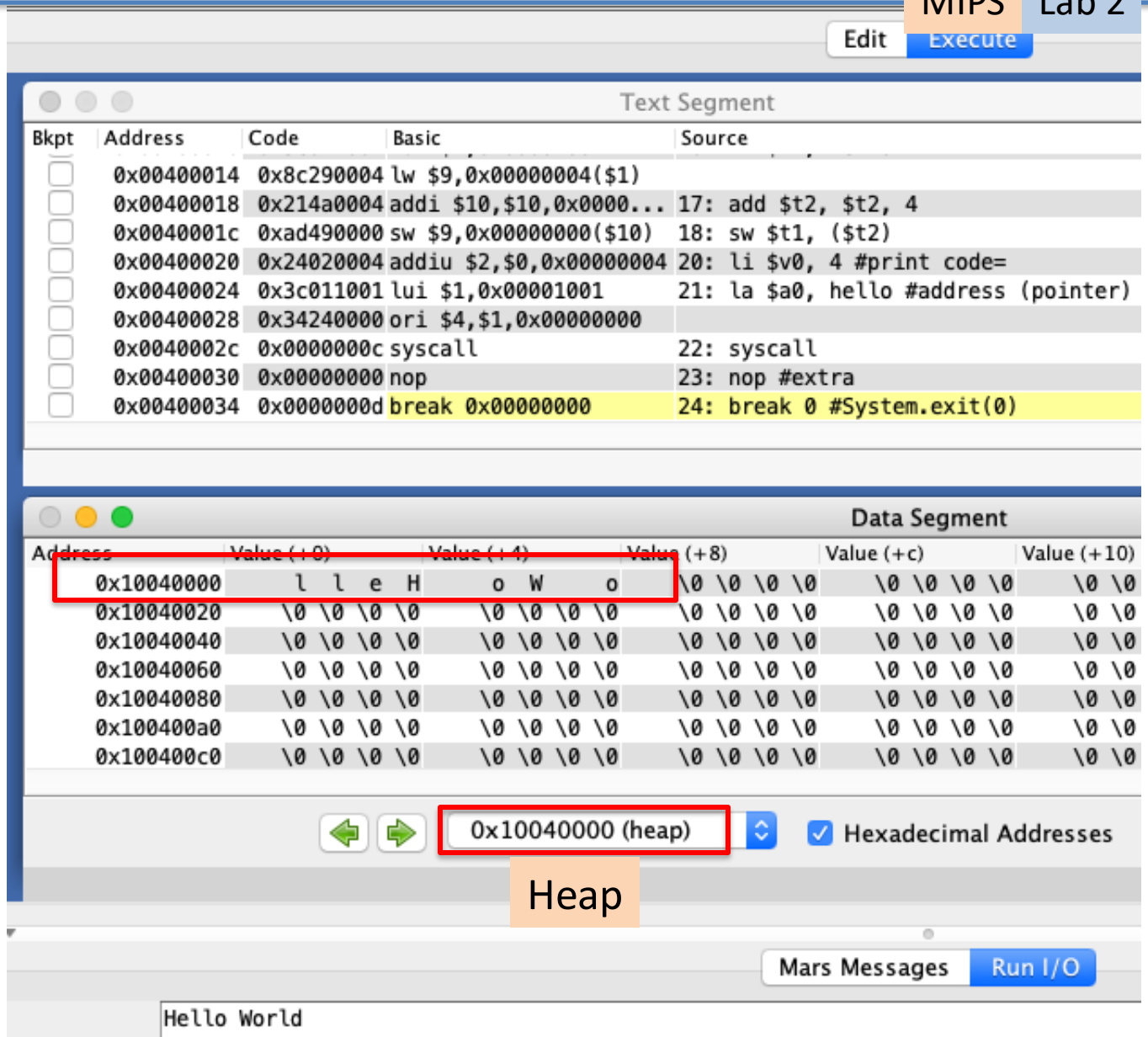
Code

Mars Messages | Run I/O

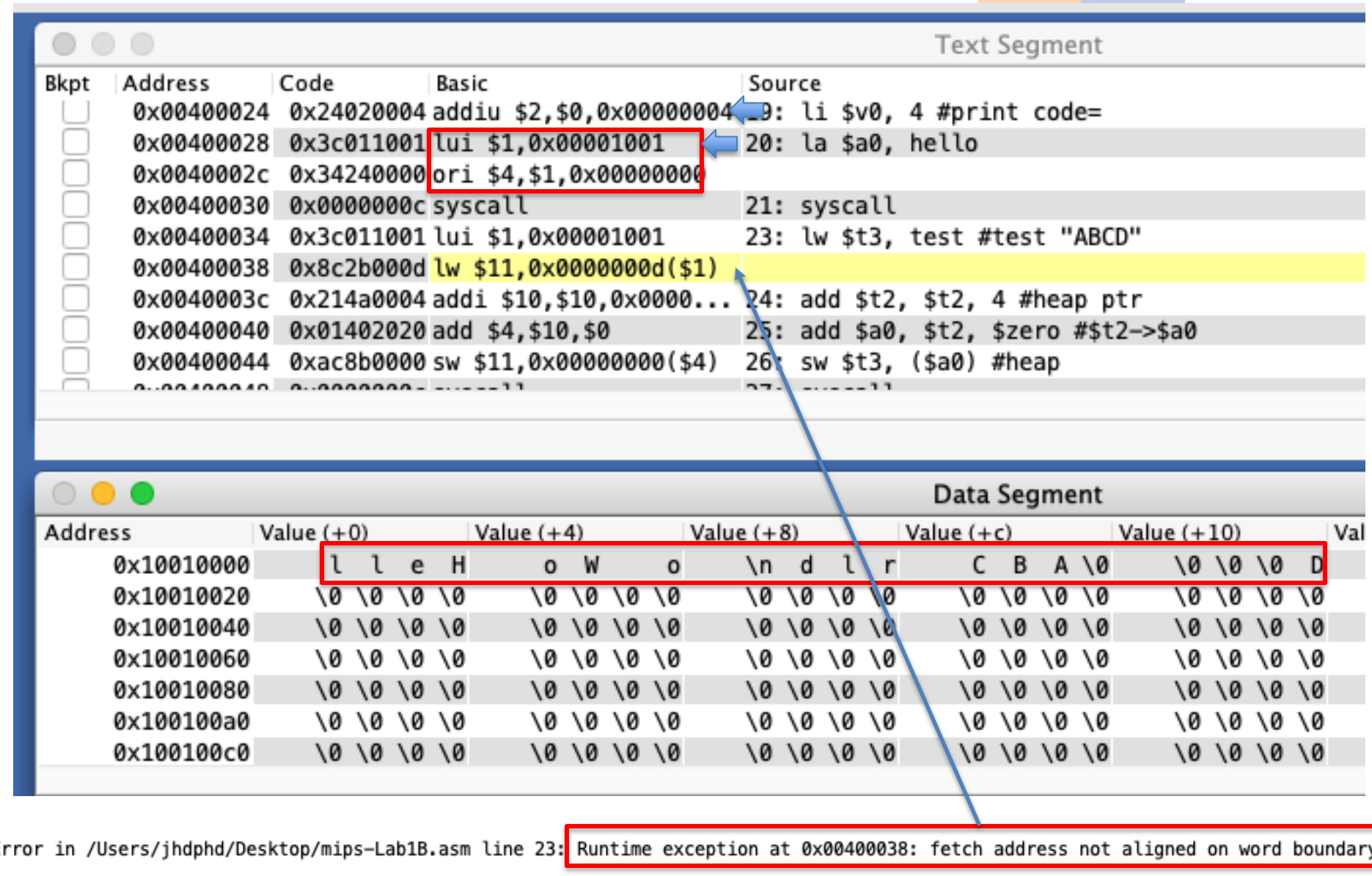
```
Hello World
```

MIPS Assembly

MIPS Lab 2



MIPS Assembly



Add a Loop

MIPS Lab 2

Patterson & Hennessy

PARTICIPATION ACTIVITY | 2.7.3: Compiling a C while loop.

Start ☐ 2x speed

Test **FIRST**

while

```
while (x == y) {
   // Loop body
}
```

```
Loop:  bne $s0, $s1, Exit
  # Loop body
  j Loop
Exit:
```

Loop: Hello

Lab 2

Java *For* Loop

```
//copy to heap (all words) using LOOP
for(int i=0; i<hello.length(); i++)
   heap[i] = charHello[i];
```

- By char

Hello loop

- By word

Assembly

loop:

bgtz

```
28  #loop for Hello
29  loop: sw $t1, ($t3) #store in memory: heap
30  addi $t2,$t2, 4 #adjust both ptrs
31  addi $t3,$t3, 4
32  lw $t1, ($t2) #next word
33  subi $t0,$t0, 1 #decr count
34  bgtz $t0,loop # !N and !Z
35  #end Loop
```

Test **LAST**

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

If-Then-Else

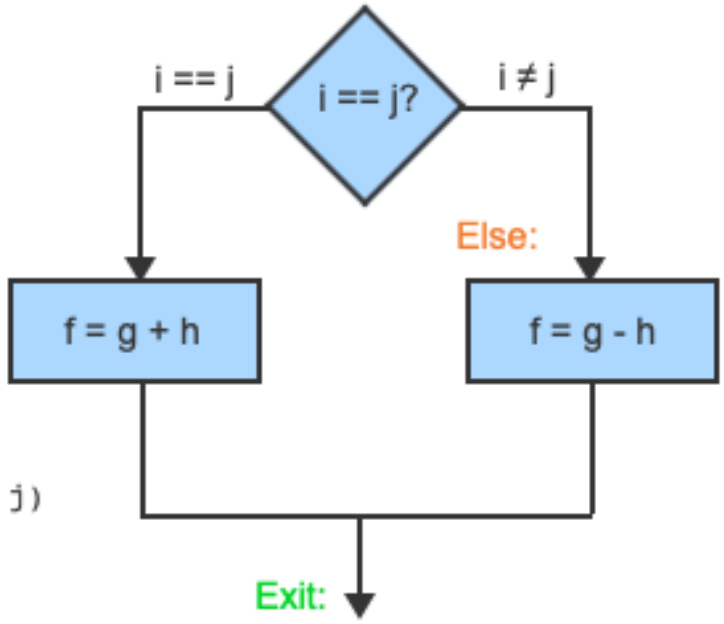
MIPS Lab 2

PARTICIPATION ACTIVITY | 2.7.1: Example of compiling if-then-else into cond Patterson & Hennessy D Figure 2.9).

Start 2x speed

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

```
bne $s3, $s4, Else   # go to Else if i ≠ j
add $s0, $s1, $s2    # f = g + h  (skipped if i ≠ j)
j  Exit              # go to Exit
Else: sub $s0, $s1, $s2   # f = g - h  (skipped if i == j)
Exit:
```



CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE

COMP122

MIPS Assembly

MIPS Lab 2

```
1  ## Lab 1B -- Hello World
2  ## by Jeff Drobman
3  ##version date: 2-4-20
4  ##add:  header, loop, macro, GUI out
5  #register map:
6  #$t1=string data, $t2=string ptr, $t3=heap ptr
7  #$t0=loop count (3)
```

```
3  ##version: 1B- Loop >10-10-19
4  .data
5  header: .asciiz  "==*==\n"
6  .align 2 #align to word
7  hello: .asciiz  "Hello World\n"
8  .align 2 #align to word
9  name:  .asciiz  "Jeff\n"
10 #define
11 .eqv heap, 0x10040000
12 .text
13 #setup Loop: ctr=$t0
14 li $t0, 3 #N=3 (not 2!)
15 #init
16 lw $t1, hello #data
17 la $t2, hello #ptr
18 la $t3, heap #0x10040000
19 #loop for Hello
20 loop: sw $t1, ($t3) #store in memory: heap
21 addi $t2,$t2, 4 #adjust ptrs
22 addi $t3,$t3, 4
23 lw $t1, ($t2) #next word
24 subi $t0,$t0, 1 #count--
25 bgtz $t0,loop
```

loop

```
26 #end Loop
27 #name -> heap
28 lw $t1, name #data
29 la $t2, name #ptr
30 sw $t1, ($t3) #heap
31 #--end write to heap
32 #--print on console using "Syscall"
33 li $v0, 4 #print str code
34 la $a0, header
35 syscall
36 la $a0, hello
37 syscall
38 la $a0, name
39 syscall
40 la $a0, heap #print entire string in heap
41 syscall
42 nop
43 li $v0, 10 #stop code
44 syscall #stop
45 #break 0 #System.exit(0)
```

Add name

?

done

```
.macro done
li $v0, 10 #stop code
syscall #stop
.end_macro
```

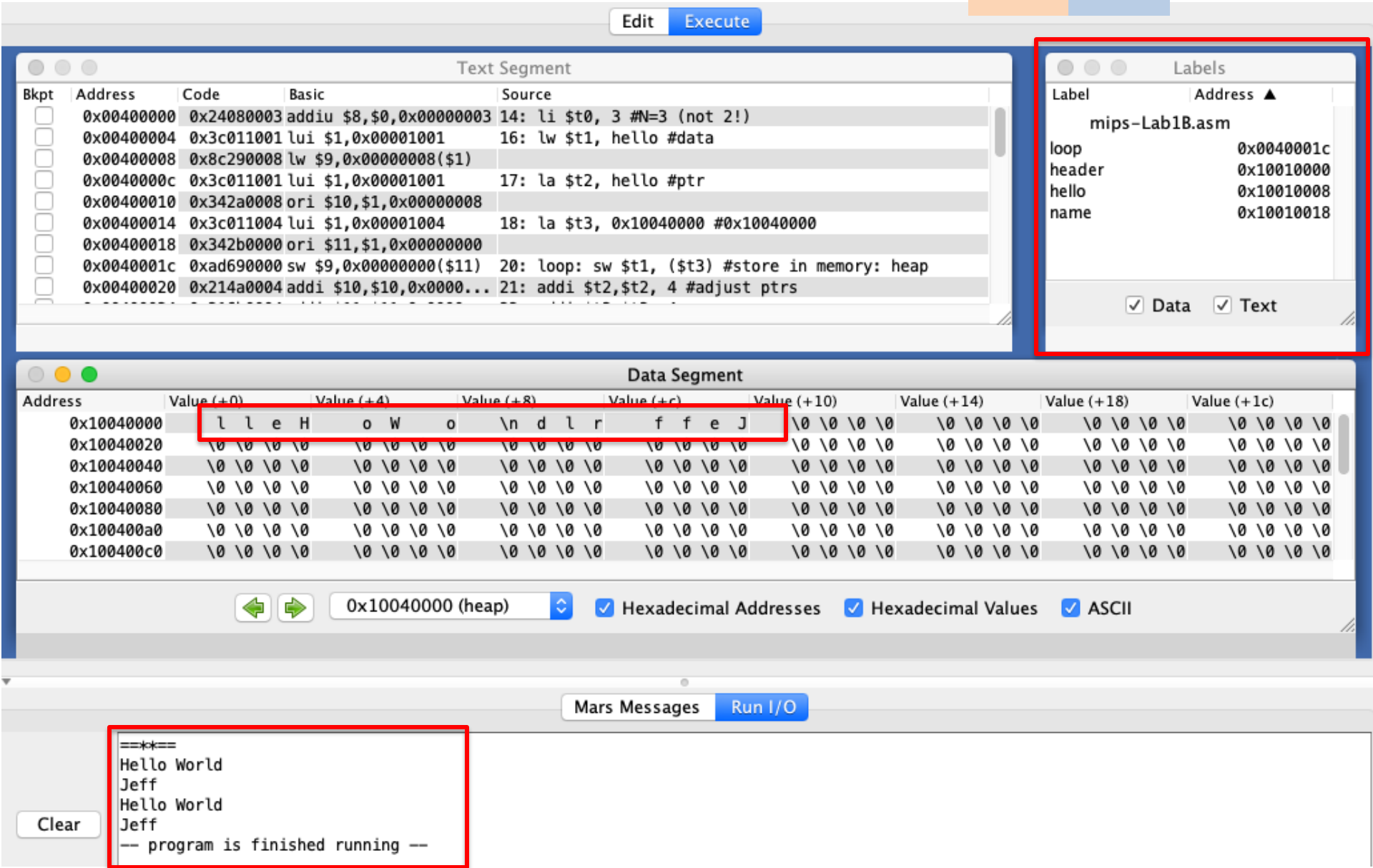
MIPS Assembly

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10040000
$v0	2	0x0000000a
$v1	3	0x00000000
$a0	4	0x10040000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x6666654a
$t2	10	0x10010018
$t3	11	0x1004000c

```
beq $t1,$t2,label       Branch if equal : Branch to statement at label's address if $t1 and $t2 are equal
bgez $t1,label          Branch if greater than or equal to zero : Branch to statement at label's address
bgezal $t1,label        Branch if greater then or equal to zero and link : If $t1 is greater than or equa
bgtz $t1,label          Branch if greater than zero : Branch to statement at label's address if $t1 is gr
blez $t1,label          Branch if less than or equal to zero : Branch to statement at label's address if
bltz $t1,label          Branch if less than zero : Branch to statement at label's address if $t1 is less
bltzal $t1,label        Branch if less than zero and link : If $t1 is less than or equal to zero, then se
bne $t1,$t2,label       Branch if not equal : Branch to statement at label's address if $t1 and $t2 are n
```

MIPS Assembly



Loop: Name

❖Simulating a *Heap*

```
11 public class Lab1Hello122 {
12 //static global DATA
13    static String hello = "Hello World!";
14    static String name = "Jeff Drobman0"; //asciiz
15    static String prompt = "Input name";
16 //main method
17    public static void main(String[] args) {
18       //simulate a "heap" = dynamic DATA segment
19       int size = 1000;
20       char[] heap = new char[size]; //reserve 1000 bytes
21       char[] charHello = hello.toCharArray();
22       char[] charName = name.toCharArray();
23       int ix = 0; //heap index
```

Loop: Name

Java | *While* Loop | ***Indefinite*** | Lab 2

```
30      //indefintite loop for var-length name (until 0 char)
31      int ix2 = 0; //new index
32      String sx = ""; //use a String
33      while(true) {
34         char chx = charName[ix2++];
35         if(chx == '0') break; //check for null 0
36         System.out.print(chx); //log char
37         heap[ix++] = chx; //store in heap
38         sx += chx; //append to heap string
39      }//end while
```

Name loop

Assembly

```
loop:
   ...
   bgtz
```

- ➢ By char — Use **LB**, **SB**
- ✓ Check for null **0**

Loop: Name


```
42        //OUTPUT
43        System.out.println(hello);//console
44        JOptionPane.showMessageDialog(null, hello);//GUI box
45        System.out.println("Heap= ");
46        for(int i=0;i<ix;i++) System.out.print(heap[i]);
47        System.out.println("<end heap>"); //clean line
```

```
 ----jGRASP exec: java Lab1Hello122
Jeff Drobman@
Hello World!
Heap=
Hello World!>Jeff Drobman<end heap>
Howdy jd
```

Case Switch (Java)

Lab 2

```
54        //CASE
55        int cNum = 1;
56        switch(cNum){
57           case 1: int x = 1;
58              break;
59           case 2: x = 2;
60              break;
61        }
```

What's this? A ***Branch Table*** Case Switch

```
106   #checks input
107   check: nop
108   beq $a1, -2, cancel
109   beq $a1, -3, nodata
110   beq $a1, -4, exceed
111   jr $ra
```

```
10    .eqv cNum, 2 #case 1
```

```
48    #check: Case= 1 or 2
49    li $s0,cNum #$s0 <-cNum
50    beq $s0,2,case2
51    #else
52    case1:
```

Case 2: Memory (Lb/Sb)


```
58  case2:
59  la $t5,24($t3) #leave a gap in heap of 24 bytes
60  sb $t1,($t5)
61  lbu $t4,($t5)
62  beqz $t4,exit #test byte0=0
63  srl $t6,$t1,8 #next byte
64  #repeat for 2nd byte, etc
65  sb $t6,1($t5)
66  lbu $t4,1($t5)
67  bnez $t4,Loop_name
68  exit:
69  #--end write to heap
```

Subroutines

Java

```
write_heap($a0,$a1, $a2,$a3);
```

Assembly

```
58  done #macro for exit
59  #end of main
60  #subroutine: write String to heap($a0..$a3)
61  write_heap: nop #sub label
62  loop: sw $a1, ($a3) #count N= $a0
63  addi $a2,$a2, 4 #adjust ptrs
64  addi $a3,$a3, 4
65  lw $a1, ($a2) #load next word
66  subi $a0,$a0, 1 #count--
67  bgtz $a0,loop
68  #return
69  jr $ra
```

loop:

bgtz

return

MIPS Assembly


```
1  ## Lab 1C -- Hello World
2  ## by Jeff Drobman
3  ##version: 1C- Loop >9-22-20
4  ##add:  sub, GUI IN: prompt+INbuf
5  #register map:
6  #$a1=string data, $a2=string ptr, $a3=heap ptr
7  #$a0=loop count (3)
8  .data
```

```
8   .data
9   header: .asciiz  "==**==\n"
10  .align 2 #align to word
11  hello: .asciiz  "Hello World\n"
12  .align 2 #align to word
13  name:  .asciiz  "Jeff D\n"
14  prompt: .asciiz  "Input string:"
15  #define
16  .eqv heap, 0x10040000
17  .eqv in_buf, 0x10040020 #input buffer
18  #macros
19  .macro done
20  li $v0, 10 #stop code
21  syscall #stop
22  .end_macro
```

MIPS Assembly

➢ Add this

MIPS Lab 2

Setup args

```
23  #code
24  .text
25  #args(4) $a0-$a3 for call "write_heap"
26  li $a0, 3 #N=3
27  lw $a1, hello #data
28  la $a2, hello #ptr
29  la $a3, heap #0x10040000
30  #call sub for hello
31  jal write_heap
32  #name -> heap
33  li $a0, 2 #N=2
34  lw $a1, name #data
35  la $a2, name #ptr
36  #$a3->heap has been updated
37  #call sub for name
38  jal write_heap
39  #--end write to heap
```

CALL

CALL

$ra <- PC

args
$a0..$a3

Sub

```
76  write_heap:
77  loop: sw $a1, ($a3) #count N= $a0
78  addi $a2,$a2, 4 #adjust ptrs
79  addi $a3,$a3, 4
80  lw $a1, ($a2) #load next word
81  subi $a0,$a0, 1 #count--
82  bgtz $a0,loop
83  #return
84  jr $ra
```

I/O

Lab 2

Output

Java

```
JOptionPane.showMessageDialog(null, msg);
```

SPIM code

Assembly

```
46  #output GUI msg
47  li $v0, 55 #GUI msg code
48  la $a0, hello
49  li $a1,1 #msg type is info
50  syscall
```

55

Input

```
//INPUT
String name = JOptionPane.showInputDialog(prompt);//GUI box
```

Assembly

```
51  #INput GUI msg
52  li $v0, 54 #GUI msg code
53  la $a0, prompt
54  la $a1, in_buf
55  li $a2,20 #max input length
56  syscall
```

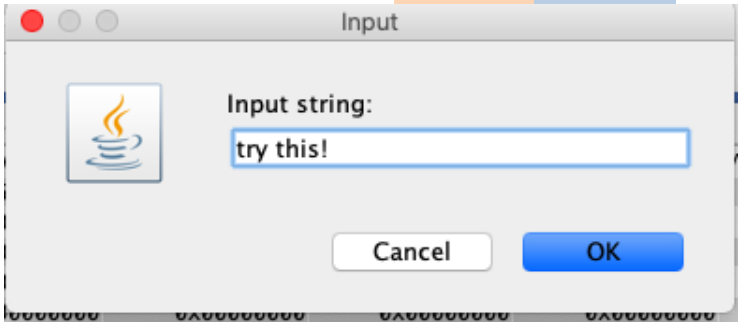
54

MIPS Assembly: GUI In

GUI Input: Syscall **54**

MIPS Lab 2

➢ Add this



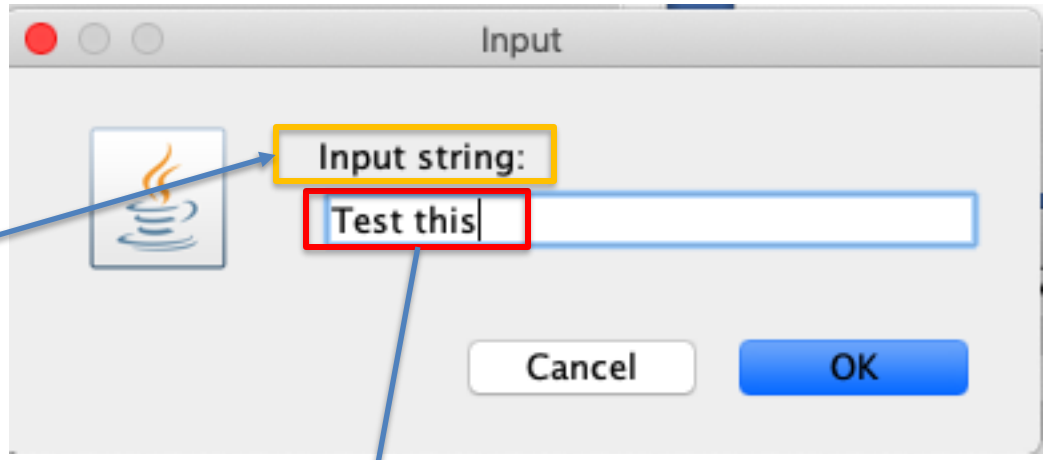
Use: Syscall **54**

InputDialogString	54	$a0 = address of null-terminated string that is the message to user $a1 = address of input buffer $a2 = maximum number of characters to read	*See Service 8 note below table* $a1 contains status value 0: OK status. Buffer contains the input string. -2: Cancel was chosen. No change to buffer. -3: OK was chosen but no data had been input into field. No change to buffer. -4: length of the input string exceeded the specified maximum. Buffer contains the maximum allowable input string plus a terminating null.

```
51  #INput GUI msg
52  li $v0, 54 #GUI msg code
53  la $a0, prompt
54  la $a1, in_buf
55  li $a2,20 #max input length
56  syscall
```

Memory Buffers: Input


```
4   .data
5   header: .asciiz  "==**==\n"
6   .align 2 #align to word
7   hello: .asciiz  "Hello World\n"
8   .align 2 #align to word
9   name:  .asciiz  "Jeff D\n"
10  prompt: .asciiz  "Input string:"
11  #define
12  .eqv heap, 0x10040000
13  .eqv in_buf, 0x10040020 #input buffer
```




```
.eqv in_buf, 0x10040020 #input buffer
```

INPUT BUFFER

54

```
51  #INput GUI msg
52  li $v0, 54 #GUI msg code
53  la $a0, prompt
54  la $a1, in_buf
55  li $a2,20 #max input length
56  syscall
```

Address	Value (+0)	Value (+4)	Value (+8)
0x10040000	l l e H	o W o	\n d l r
0x10040020	t s e T	i h t	\0 \0 \n s
0x10040040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10040060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10040080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100400a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100400c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100400e0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

0x10040000 (heap)

Check for Error

What's this? A ***Branch Table***

```
106   #checks input
107   check: #Sub entry point
108   beq $a1, -2, cancel
109   beq $a1, -3, nodata
110   beq $a1, -4, exceed
111   jr $ra
```

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

MIPS Code Puzzle

Fork this | Run | Lang

main.S

```
50  sw $t4, 0($t1)
51  li $t4, 0b00000000 # turn off all leds
52  sw $t4, 0($t1)
53
54  switch_3:
55  li $t4, 0b00000001 # turn on first led
56  # *t1 = t4
57  sw $t4, 0($t1)
58  shift_t4: #loop to make cycle through all 8 LEDs
59  sll $t4, $t4, 1
60  sw $t4, 0($t1)
61  beq $t4, $0, end # put END condition FIRST to avoid looping infinitely
62  jump_shift_t4:
63  j shift_t4
64
65  end:
66  j end
67  nop
```

While(true)

SLEEP/WAIT

Lab

LAB 3

Radix Conversion

Lab 3

Requirements

- Input 3 digits each (assign)
  1. Hex number
  2. Decimal number
  3. BCD (un-packed) number
- Process
  1. Convert Hex → Decimal (as subroutine)
- Output
  - GUI message (lab title)
  - Console
    1. Hex to Dec Conversion
- Code
  1. Use “done” macro
  2. Convert all I/O code into subroutines
  3. Complete “check” subroutine (add branch table)

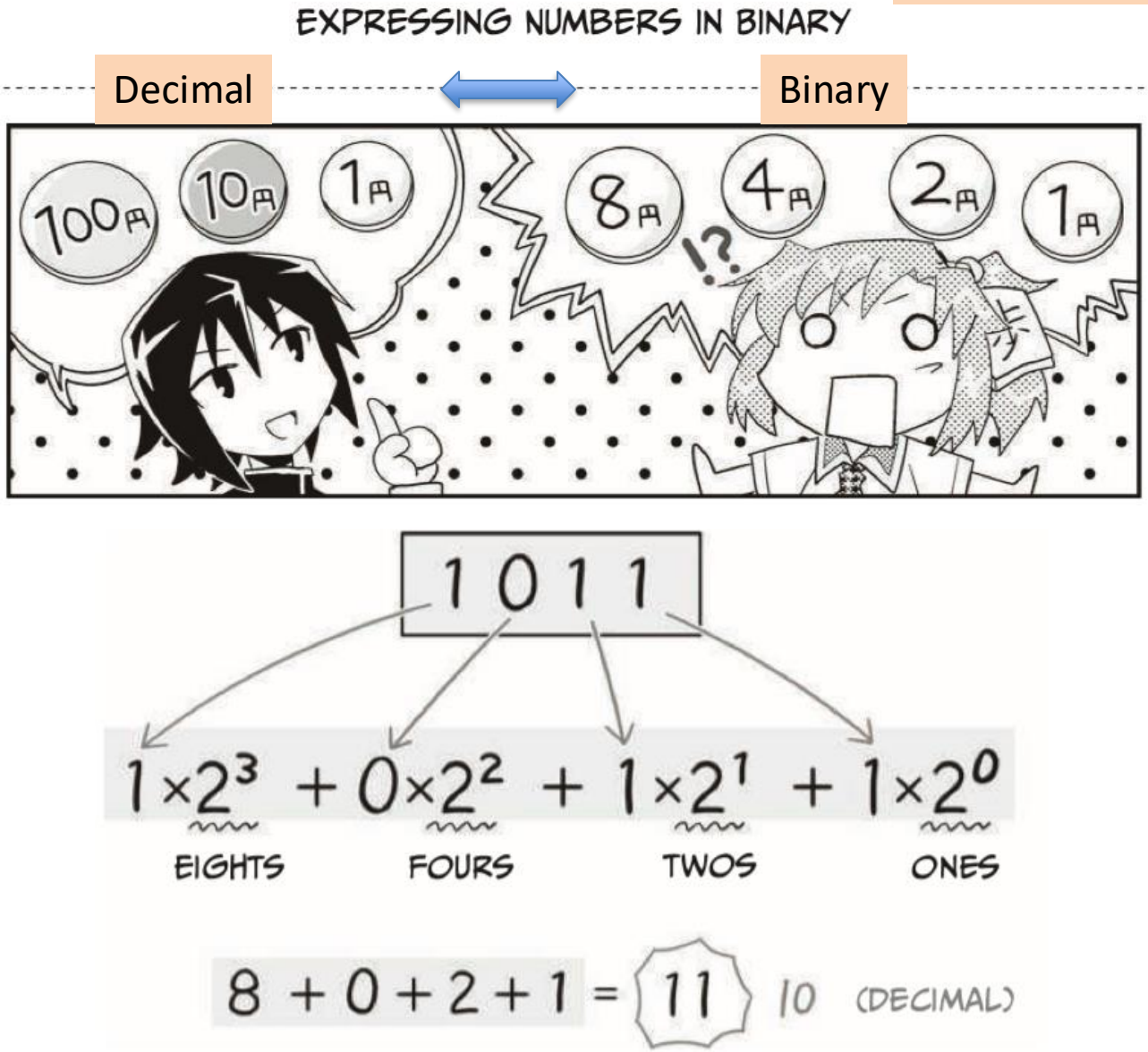
Lab 3

Construction

- I/O
  - GUI
    1. Create subroutines for each GUI (54, 55, 59)
    2. Integrate "check" sub into Input (54)
  - Console
    1. Create subroutines for each print (1, 4)
- Strings
  1. Add *title* string
  2. Keep your prompt & error strings from Lab 2
  3. Delete other strings (Hello, name)
- Eqv
  1. Keep pointer to heap
  2. Keep pointer to input buffer (for 54)

Radix: Binary to Decimal

Manga Guide



Hex to Decimal

256xd2 + 16xd1 +d0

.data

Process (code)

Setup Data

```
5   #register map:
6   #$t0=calc result, $t1=hexu data, $t2=val16
7   ##
--
13   .eqv in_buf, 0x10040020
14   .eqv heapHi, 0x1004 #heap=0x10040000

16  .data
17  #setup
18  hex2dec: .ascii "****" #reserve word & mark
19  val16: .word 256, 16, 1 #int[] val16={256,16,1}
20  #hex to dec: gen output string
21  hexu: .word 0x1050c #unpacked
22  hexp: .ascii "15c" #packed; start of output string
23  hexStr: .asciiz " in hex = in decimal:  " #word align
24  title: .asciiz "Lab 3: convert Hex to Dec"
25  prompt: .asciiz "Input a new decimal number"
26  .align 2 #align to word
27  ovfl: .asciiz "\nError: max length exceeded!\n"
28  error: .asciiz "\nError: canceled or empty"
```

.text Main

Process (code)

Main

Console out

GUI in

```
34  .text
35  jal GUI_msg
36  #convert hex to dec (use unrolled loop)
37  sw $0,hex2dec #init to 0
38  #perform conversion
39  jal convert
40
41  #print it all
42  #console
43  jal print_str
44  jal print_int
45  #GUI new dec input
46  jal GUI_in
47  #stored in "in_buf"
48  _done
49  ##END of Main##
```

App Sub


```
46  #app specific sub
47  convert:
48  lb $t1,hexu+? #d2
49  lw $t2,val16 #v2, word!
50  mult  $t1,$t2 #[Hi,Lo]= d2*v2
51  lb $t1,hexu+? #d1 (next byte)
52  lb $t2,val16+? #v1
53  madd $t1,$t2 #[Hi,Lo]+= d1*v1
54  lb $t1,hexu+? #d0
55  lb $t2,val16+? #v0
56  madd $t1,$t2 #[Hi,Lo]=d0*v0
57  mflo $t0 #dec value
58  sw $t0,hex2dec
59  jr $ra
```

I/O Subs


```
61  #I/O subs
62  print_str:
63  li $v0,4 #print string
64  la $a0,hexp
65  syscall #print output string
66  jr $ra
67  print_int:
68  li $v0,1 #print int
69  move $a0,$t0 ##a0=dec value
70  syscall #print dec value
71  jr $ra
72  GUI_msg:
73  li $v0, 55 #GUI msg code
74  la $a0, title
75  li $a1,1 #msg type is info
76  syscall
77  jr $ra
```

Java Parse Hex

Quora

Gourav Aggarwal · Follow
B.Tech from ABES Engineering College, Ghaziabad (Graduated 2023) · Feb 10

How do you convert a hexadecimal number to decimal in Java?

```
1  String hexadecimal = "A1F";
2  int decimal = Integer.parseInt(hexadecimal, 16);
3  System.out.println("Hexadecimal number: " +
   hexadecimal);
4  System.out.println("Decimal equivalent: " + decimal);
```

```
Also,
String bit = "01101001";
int decimal = Integer.parseInt(bit, 2);
```

Java Parse Hex


```
7  // imports
8  import java.util.Scanner;
9  // **main class**
10 public class Bin2hex {
11 //main method
12    public static void main(String[] args) {
13       int[] dec = {0,1,2,3,4,5,6,7,8,9};
14       char[] hex = {'A','B','C','D','E','F'};
15       int ascii;
16       boolean error = false;
17       Scanner input = new Scanner(System.in);
18       //start input loop
19       while (!error) {
20       String digit = input.next();
21       if (digit.equalsIgnoreCase("Q")) {
22          System.out.println("You quit");
23          System.exit(0);}
24       int bin = Integer.valueOf(digit,2); //vs. parseInt
25       if(bin<0 || bin>15) { //check
26          System.out.println("error: input out of range");
27          error = true;
28          continue;}
29       System.out.print("converted to Hex / ASCII: ");
```

```
----jGRASP exec: java Bin2hex
1111
converted to Hex / ASCII: F / 70
0010
converted to Hex / ASCII: 2 / 50
```

Decimal to Binary

A bit harder

Algorithm

1. Find the largest power of 2 that is < N (decimal number)
2. Set that bit to 1
3. Subtract 2^n from N
4. Repeat 1-3

Examples

Chunk into Hex

1. N = 10 → **2^3**=8 → 10-8=2 → **2^1** → **1**0**1**0
2. N = 50 → **2^5**=32 → 18 → **2^4**=**16** → 2 → **2^1** → **11** 00**1**0
3. N = 100 → **2^6**=64 → 36 → **2^5**=32 → 4 → **2^2** → **11**0 0**1**00

3x16+2=**50**

6x16+4=**100**

Lab 3 Code errors


```
27  .text
28  #convert hex to dec (use unrolled loop)
29  sw $0,hex2dec #init to 0
30  mult $0,$0 #clear Hi,Lo
31  lb $t1,hexu #d0
32  lb $t2,val16 #v0
33  madd $t1,$t2 #[Hi,Lo]+= $t1*$t2
34  lb $t1,hexu+1 #d1 (next byte)
35  lb $t2,val16+4 #v1
36  madd $t1,$t2
37  lb $t1,hexu+2 #d2
38  lb $t2,val16+8 #v2
39  madd $t1,$t2 #[Hi,Lo]=final dec value
40  mflo $t0 #dec value
41  sw $t0,hex2dec
42  li $v0,4 #print string
43  la $a0,hexp
44  syscall #print output string
45  li $v0,1 #print int
46  move $a0,$t0 ##a0=dec value
47  syscall #print dec value
48  #convert dec to hex
49
50  #convert bcd to dec
51
52  done
```

- Unrolled loop
- For 3 digits

$t0 = decimal value

Print decimal value in string

Fixed Lab 3 Code

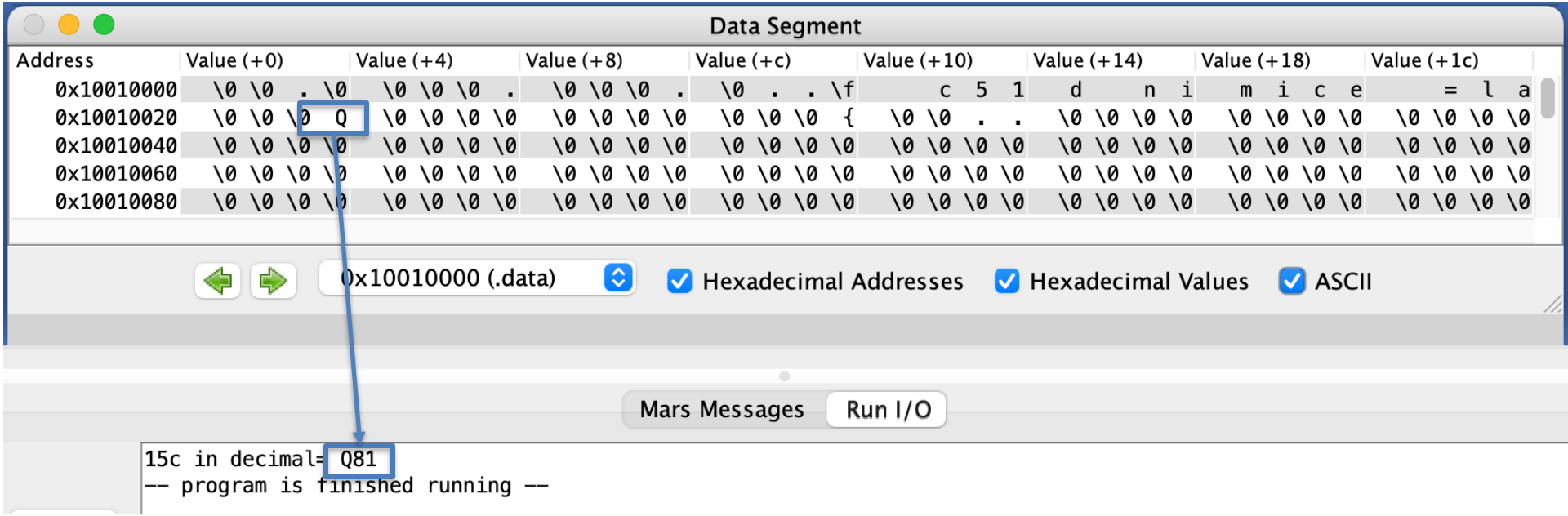

```
27  .text
28  #convert hex to dec (use unrolled loop)
29  sw $0,hex2dec #init to 0
30  mult $0,$0 #clear Hi,Lo
31  lb $t1,hexu+2 #d2
32  lw $t2,val16 #v0, word!
33  madd $t1,$t2 #[Hi,Lo]+= $t1*$t2
34  lb $t1,hexu+1 #d1 (next byte)
35  lb $t2,val16+4 #v1
36  madd $t1,$t2
37  lb $t1,hexu #d0
38  lb $t2,val16+8 #v2
39  madd $t1,$t2 #[Hi,Lo]=fina
40  mflo $t0 #dec value
41  sw $t0,hex2dec
42  li $v0,4 #print string
43  la $a0,hexp
44  syscall #print output string
45  li $v0,1 #print int
46  move $a0,$t0 ##a0=dec value
47  syscall #print dec value
48  #convert dec to hex
49
50  #convert bcd to dec
```

- Unrolled loop
- For 3 digits

$t0 = decimal value

Print decimal value in string

Lab 3 Output



Wrong answer: 0x51 = 81 decimal (ASCII code for "Q")
Correct answer: 0x15c = 256+80+12 = 348 decimal

Lab

LAB 4 MIPS

Fibonacci Sequence

Lab 4

Requirements

- Input
  1. Set N >= 15
  2. Create an array *fibs[N]*
- Process
  1. Generate a Fibonacci sequence for N
  2. Store in array *fibs*
  3. Print 8 numbers per line
- Output
  1. Fibonacci sequence for N (Console)
  2. Print 8 numbers per line

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Lab 4: Fibs in Arrays

Hennessy & Patterson

PARTICIPATION ACTIVITY 2.3.5: Example of compiling an assignment when an operand is in memory.

Start ☐ 2x speed

```
g = h + A[8];
```

$s1	60	g
$s2	35	h
$s3	5000	A's base addr
$t0	25	

Registers

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

8 + 5000

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

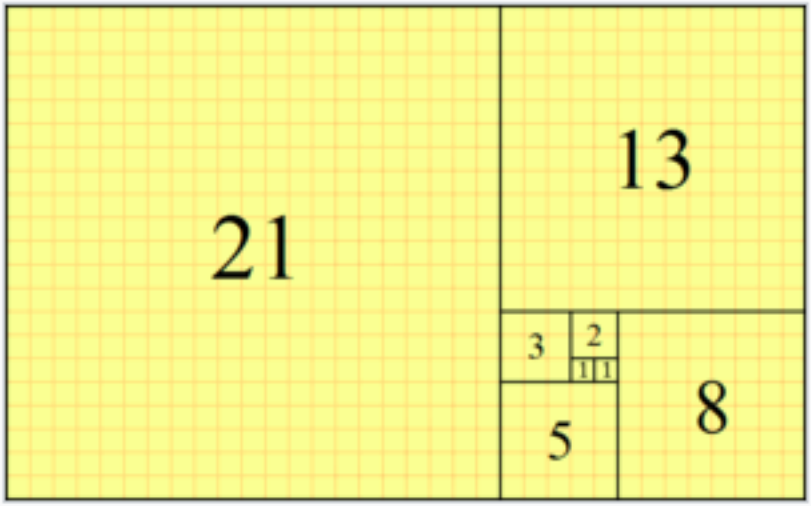
35 + 25

5009		(100 words)
5008	25	A[8]
5007		
5006		
5005		offset
5004		
5003		
5002		
5001		
5000		A
4999		

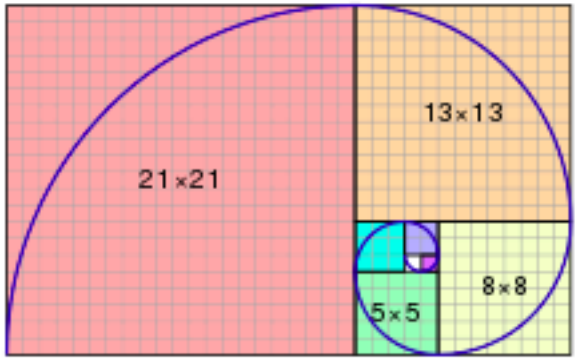
Memory

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Fibonacci Sequence



A tiling with squares whose side lengths are successive Fibonacci numbers: 1, 1, 2, 3, 5, 8, 13 and 21.



The Fibonacci spiral: an approximation of the golden spiral created by drawing circular arcs connecting the opposite corners of squares in the Fibonacci tiling; (see preceding image)

The first 21 Fibonacci numbers F_n are:[2]

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

Fibonacci in Java

Lab 4

```
public class Lab2Fibs122 {
   public static void main(String[] args) {
      int N = 12;
      int[] F = new int[N];
      F[0]=1; F[1]=1;
      System.out.print("Fibs: " +F[0] +" " +F[1] +" ");
      for(int i=2; i<N; i++) {
         F[i] = F[i-1] + F[i-2];
         System.out.print(F[i] +" ");
         }//end for
      System.out.println("\n-----");
      printit(F);//call printit
   }//end main
   //printit sub
   static void printit(int[] arr) {
      System.out.print("Fibs: ");
      for(int x: arr)
         System.out.print(x +" ");
      System.out.println();
   }//end printit
}//end class
```

Print inline

Print sub

```
----jGRASP exec: java Lab2Fibs122
Fibs: 1 1 2 3 5 8 13 21 34 55 89 144
-----
Fibs: 1 1 2 3 5 8 13 21 34 55 89 144
```

Fibonacci

MIPS

Compute first twelve ***Fibonacci*** numbers and put in array, then print

```
        .data
fibs:   .word   0 : 12      # "array" of 12 words to contain fib values
size:   .word   12          # size of "array"
        .text
        la    $t0, fibs     # load address of array
        la    $t5, size     # load address of size variable
        lw    $t5, 0($t5)   # load array size
        li    $t2, 1        # 1 is first and second Fib. number
        add.d $f0, $f2, $f4 ?
        sw    $t2, 0($t0)   # F[0] = 1
        sw    $t2, 4($t0)   # F[1] = F[0] = 1
        addi  $t1, $t5, -2  # Counter for loop, will execute (size-2) times
loop:   lw    $t3, 0($t0)   # Get value from array F[n]
        lw    $t4, 4($t0)   # Get value from array F[n+1]
        add   $t2, $t3, $t4 # $t2 = F[n] + F[n+1]
        sw    $t2, 8($t0)   # Store F[n+2] = F[n] + F[n+1] in array
        addi  $t0, $t0, 4   # increment address of Fib. number source
        addi  $t1, $t1, -1  # decrement loop counter
        bgtz  $t1, loop     # repeat if not finished yet.
        la    $a0, fibs     # first argument for print (array)
        add   $a1, $zero, $t5 # second argument for print (size)
        jal   print         # call print routine.
        li    $v0, 10       # system call for exit
        syscall             # we are out of here.
```

Loop

```
1
1
2
3
5
8
13
21
34
55
89
144
```

Fibonacci

MIPS

Compute first twelve ***Fibonacci*** numbers and put in array, then print

```
#########  routine to print the numbers on one line.

        .data
space:.asciiz   " "             # space to insert between numbers
head: .asciiz   "The Fibonacci numbers are:\n"
        .text
print:add   $t0, $zero, $a0     # starting address of array
        add   $t1, $zero, $a1   # initialize loop counter to array size
        la    $a0, head         # load address of print heading
        li    $v0, 4            # specify Print String service
        syscall                 # print heading
out:    lw    $a0, 0($t0)       # load fibonacci number for syscall
        li    $v0, 1            # specify Print Integer service
        syscall                 # print fibonacci number
        la    $a0, space        # load address of spacer for syscall
        li    $v0, 4            # specify Print String service
        syscall                 # output string
        addi $t0, $t0, 4        # increment address
        addi $t1, $t1, -1       # decrement loop counter
        bgtz $t1, out           # repeat if not finished
        jr    $ra               # return
```

Loop

```
1
1
2
3
5
8
13
21
34
55
89
144
```

ALU Ops

MARS


```
add $t1,$t2,$t3          Addition with overflow : set $t1 to ($t2 plus $t3)
add.d $f2,$f4,$f6        Floating point addition double precision : Set $f2 to double-precision floating p
add.s $f0,$f1,$f3        Floating point addition single precision : Set $f0 to single-precision floating p
addi $t1,$t2,-100        Addition immediate with overflow : set $t1 to ($t2 plus signed 16-bit immediate)
addiu $t1,$t2,-100       Addition immediate unsigned without overflow : set $t1 to ($t2 plus signed 16-bit
addu $t1,$t2,$t3         Addition unsigned without overflow : set $t1 to ($t2 plus $t3), no overflow
and $t1,$t2,$t3          Bitwise AND : Set $t1 to bitwise AND of $t2 and $t3
andi $t1,$t2,100         Bitwise AND immediate : Set $t1 to bitwise AND of $t2 and zero-extended 16-bit im
```

Fibonacci in MIPS

MIPS


```
1  ## Lab 2 -- Fibonacci
2  ## by Jeff Drobman
3  ##version: 1.0 >11-21-19
4  .data
5  header: .asciiz  "Fibonacci sequence by Jeff D\n"
6  .align 2
7  fibs: .word  0:12
8  .align 2
9  size:  .word  12
10 space:  .word  0x20 #space
11 #define
12 #.eqv heap, 0x10040000 -not used
13 #macros
14 .macro done
15 li $v0, 10 #stop code
16 syscall #stop
17 .end_macro
18 #code
19 .text
20 #set up pointers
21 la $t0, fibs #ptr
22 lw $t5, size #final
23 subu $t1,$t5,2 #counter
24 #init 1st 2 numbers (1, 1)
25 li $t2,1
26 li $t3,1
27 sd $t2,($t0) #both 1's
28 addi $t0,$t0,8 #incr ptr
29 loop_main:
30 add $t4,$t2,$t3 #next fib
```

```
Fibonacci sequence by Jeff D
1 1 2 3 5 8 13 21 34 55 89 144
-- program is finished running --
```

```
1
1
2
3
5
8
13
21
34
55
89
144
```

pseudo

```
sd $t1,($t2)
sd $t1,-100($t2)
sd $t1,100000
sd $t1,100000($t2)
sd $t1,label
sd $t1,label($t2)
sd $t1,label+100000
sd $t1,label+100000($t2)
```

Fibonacci: Dr Jeff

MIPS

Loop

```
31  sw $t4,($t0)
32  addi $t0,$t0,4 #incr ptr
33  move $t2,$t3
34  move $t3,$t4
35  subi $t1,$t1,1 #counter
36  bgtz $t1,loop_main
37  #call sub for print
38  jal print
39  done #macro for exit
40  #---end of main---
41  #subroutine: print($t0=ptr, $t5=size)
42  print: nop #sub label
43  #print header
44  li $v0, 4
45  la $a0, header
46  syscall
47  #print fibs
48  la $t0, fibs #ptr
49  loop_prt:
50  li $v0, 1 #print int code
51  lw $a0, ($t0)
52  syscall
53  li $v0, 11 #print char code
54  lw $a0, space
55  syscall
56  addi $t0,$t0,4 #incr ptr
57  subi $t5,$t5, 1 #count--
58  bgtz $t5,loop_prt
59  #return
60  jr $ra
```

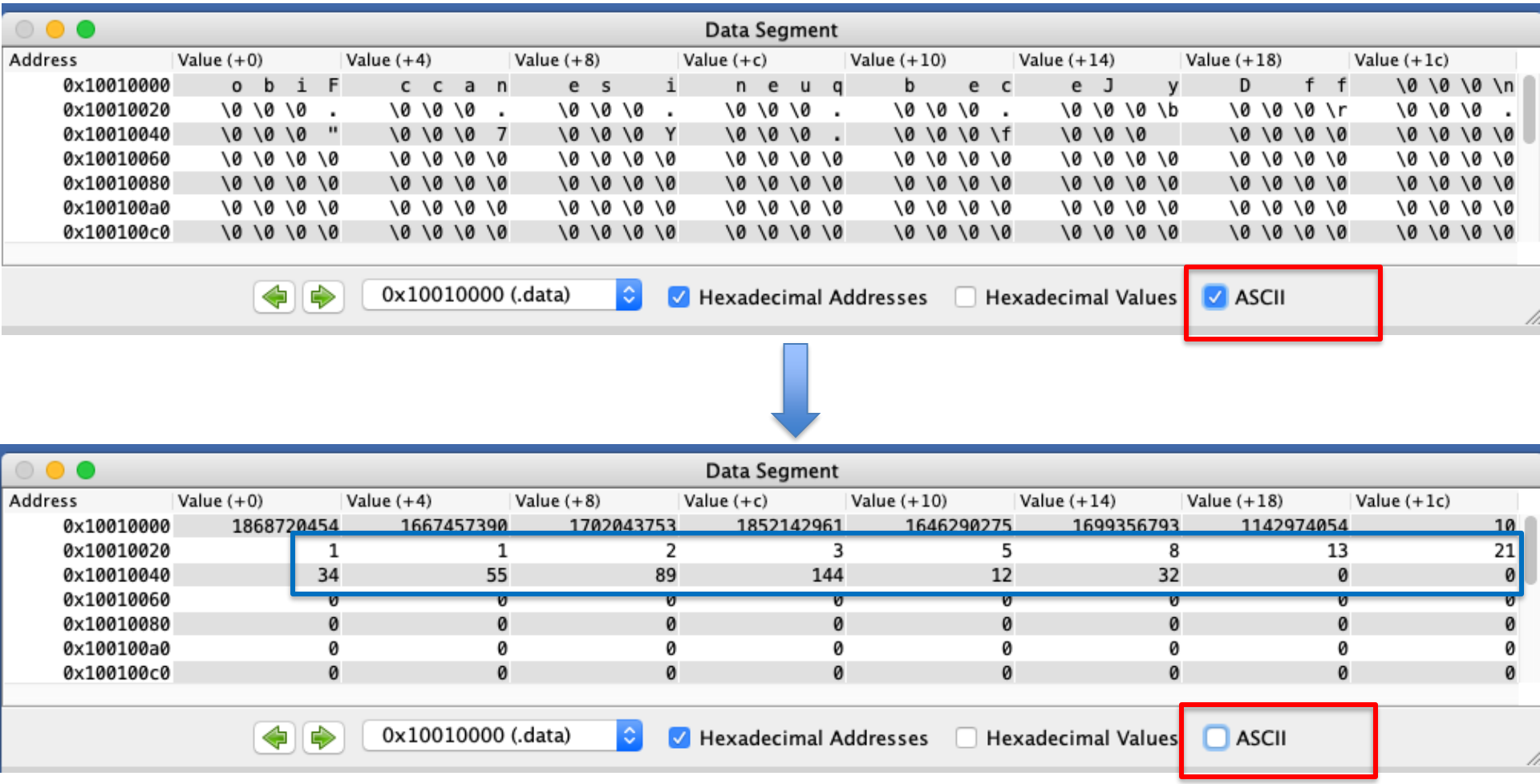
```
Fibonacci sequence by Jeff D
1 1 2 3 5 8 13 21 34 55 89 144
-- program is finished running --
```

1
1
2
3
5
8
13
21
34
55
89
144

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Fibonacci: Dr Jeff

MIPS



CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Lab 4: 10 per Line


```
1  ## Lab 6 -- Fibonacci
2  ## by Jeff Drobman
3  ##version: 1.0 >10-3-23
4  #register map
5  #$s0= size
6  #$t0= array ptr , #t1/t5=loop ctr, $t2=F1 , $
7  .data
8  .eqv size, 28 #variable init
9  header: .asciiz  "Fibonacci sequence by Jeff
```

Line: 16 Column: 9 Show Line Numbers

Mars Messages | Run I/O

```
Fibonacci sequence by Jeff D
1  1  2  3  5  8  13  21  34  55
89  144  233  377  610  987  1597  2584  4181  6765
10946  17711  28657  46368  75025  121393  196418  317811
-- program is finished running --
```

Clear

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

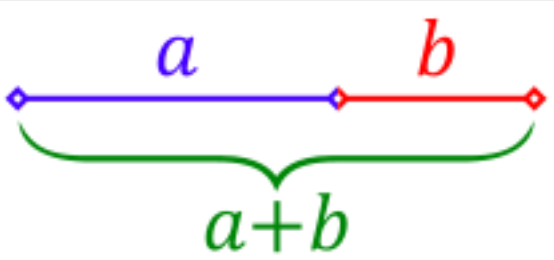
Fibonacci—Golden Ratio

Golden ratio

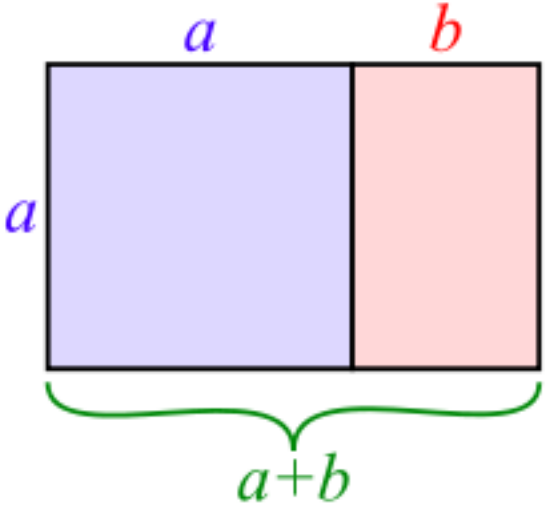
From Wikipedia, the free encyclopedia

$$\frac{a+b}{a} = \frac{a}{b} \overset{\text{def}}{=} \varphi,$$

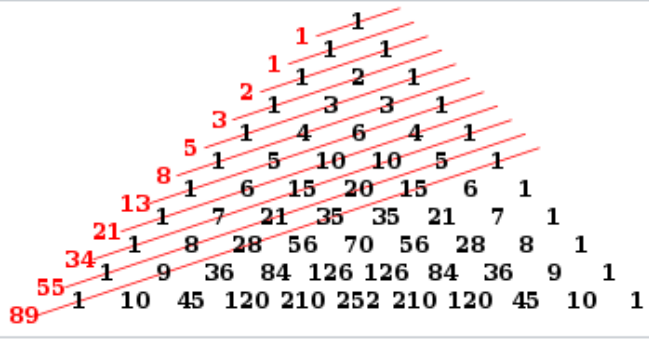
$$\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887\ldots$$



Line segments in the golden ratio



A golden rectangle with longer side ***a*** and shorter side ***b***, when placed adjacent to a square with sides of length ***a***, will produce a similar golden rectangle with longer side ***a + b*** and shorter side ***a***. This illustrates the relationship $\frac{a+b}{a} = \frac{a}{b} \equiv \varphi.$



The Fibonacci numbers are the sums of the "shallow" diagonals (shown in red) of Pascal's triangle.

Fibonacci in Java


```
 ----jGRASP exec: java Fibs
Fibs as gen:
1  1  2  3  5  8  13  21  34  55  89  144  233
377  610  987  1597  2584  4181  6765  10946  17711  28657  46368  75025
121393  196418  317811  514229  832040
final 3: 317811  514229  832040
-----
Fibs from array: 30
1  1  2  3  5  8  13  21  34  55  89  144  233
377  610  987  1597  2584  4181  6765  10946  17711  28657  46368  75025
121393  196418  317811  514229  832040
-----
Golden ratio convergence:
1.0  2.0  1.5  1.6666666666666667  1.6  1.625  1.6153846153846154  1.61904761904
29 1.6180339887482036  28 1.6180339887543225  27 1.618033988738303

 ----jGRASP: operation complete.
```

Golden Ratio

Fibonacci in Java


```
 7 public class Fibs {
 8    static String spc2 = "   ";
 9    static int Ngen = 30, Nprt = 30, line = 12;
10    public static void main(String[] args) {
11       long[] F = new long[Ngen];
12       //call Generate method
13       genFibs(Ngen,F);
14       //call Print method
15       printit(Nprt,F);
16       //call Golden method
17       golden(Ngen,F);
18    }//end main
```

Fibonacci in Java

Lab 4

```
20    //Generate method
21    static void genFibs(int N,long[] F) {
22       F[0]=1; F[1]=1;
23       System.out.println("Fibs as gen: ");
24       System.out.print(F[0] +spc2 +F[1] +spc2);
25       for(int i=2; i<N; i++) {
26          F[i] = F[i-1] + F[i-2];
27          if(i<=Nprt) System.out.print(F[i] +spc2);//max limit
28          if(i>0 && i<=Nprt && i%line==0) System.out.println();//15 per line
29          }//end for
30       //System.out.println("\nfinal 3: " +F[N-3] +spc2 +F[N-2] +spc2 +F[N-1]);
31       System.out.println("-----");
32    }//end Gen
33
34    //Print method
35    static void printit(int N, long[] F) {
36       System.out.println("Fibs from array: " +N);
37       for(int i=0;i<N;i++) {
38          if(i<=Nprt) System.out.print(F[i] +spc2);//max limit
39          if(i>0 && i%line==0) System.out.println();//15 per line
40          }
41       System.out.println("\n-----");
42    }//end print
```

Print inline

Print sub

Fibonacci in Java


```
44     //Golden method
45     static void golden(int N, long[] F) {
46        int M = 13;//1st 10 + last 3
47        double[] gold = new double [M];
48        System.out.println("Golden ratio convergence: ");
49        for(int i=0;i<10;i++) {
50           gold[i] = F[i+1]/(double)F[i];
51           System.out.print(gold[i] +spc2);}
52           System.out.println();
53        for(int i=0;i<3;i++) {
54           gold[i+3] = F[N-1-i]/(double)F[N-2-i];
55        System.out.print(N-1-i +" " +gold[i+3] +spc2);}
56        System.out.println();
57     }//end Golden
58  }//end class
```

Array Swap


```
//C conversion to MIPS
int N = 5; // some value
int a[ ] = {1,2,3,4,5}
int b[ ]= {6,7,8,9,10}
for (int i=0; I<N; I++) {//swap
   int temp = a[i];
   a[i] = b[i]
   b[i] = temp;
}
```

```
1  #MIPS by Jeff Drobman, 4-6-21
2  #reg map: $t0=count, $t1=*a, $t2=*b, $t3=temp
3  .eqv N, 5
4  .data
5  a:   .word 1,2,3,4,5
6  spacer: .ascii "****&&&&****" #3 words
7  b:   .word 6,7,8,9,10
8  .text
9  li $t0,N #loop counter
10 la $t1,a
11 la $t2,b
12
13 Loop: #for(I=N; I>0; I--)
14 lw $t3,($t1) #temp=a
15 lw $t4,($t2) #t4=b
16 sw $t4,($t1) #a=b
17 sw $t3,($t2) #b=temp
18 #update ptrs
19 addiu $t1,$t1,4
20 addiu $t2,$t2,4
21 #loop end
22 subi $t0,$t0,1
23 bgtz $t0,Loop #end Loop
24 #end of program
```

Array Swap

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	1	2	3	4	5	707406378	640034342	707406378
0x10010020	6	7	8	9	10	0	0	0

```
4   .data
5   a:   .word 1,2,3,4,5
6   spacer: .ascii "****&&&&****" #3 words
7   b:   .word 6,7,8,9,10
8   .text
9   li $t0,N #loop counter
10  la $t1,a
11  la $t2,b
12
13  Loop: #for(I=N; I>0; I--)
14  lw $t3,($t1) #temp=a
15  lw $t4,($t2) #t4=b
16  sw $t4,($t1) #a=b
17  sw $t3,($t2) #b=temp
18  #update ptrs
19  addiu $t1,$t1,4
20  addiu $t2,$t2,4
21  #loop end
22  subi $t0,$t0,1
23  bgtz $t0,Loop #end Loop
```

Swapped

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	6	7	8	9	10	707406378	640034342	707406378
0x10010020	1	2	3	4	5	0	0	0

Lab

LAB 5 MIPS

Factorials

Lab 5

Requirements

- Input
  1. Set N = 20
  2. Create an array *facts[N]*
- Process
  1. Generate a *factorial* sequence for N
  2. Store in array *facts*
  3. Use 32-bit integers
  4. Detect *overflow* & signal: Check "hi" !=0
- Output
  1. Count and header
  2. *Factorial* sequence (Console)
  3. Print 5-8 numbers per line
  4. Signal when overflow occurs

Nesting Subs

- If you want to call a subroutine from within a subroutine
- You need to save/restore the first ***return address***
  1. Use a **register** (move)
  2. Use **memory** (lw/sw)
  3. Use the **stack**: lw $t1,($sp)

Factorials

Lab 5

Java: non-*recursive*

```
5  import java.util.Scanner;
6
7  public class Facts {
8     static String spc2 = "   ";
9     static int Ngen = 30, Nprt = 30, line = 12;
10    public static void main(String[] args) {
11       long[] F = new long[Ngen];
12       //get Ngen
13       Scanner input = new Scanner(System.in);
14       System.out.println("Input N: ");
15       Ngen = input.nextInt();
16       //call Generate method
17       genFacts(Ngen,F);
18       //call Print method
19       if(Nprt>Ngen) Nprt = Ngen;//limit
20       printit(Nprt,F);
21    }//end main
22
23    //Generate method
24    static void genFacts(int N,long[] F) {
25       F[0]=1;
26       System.out.println("Factorials as gen: ");
27       for(int i=1; i<N; i++) {
28          F[i] = F[i-1] *i;
29          if(i<=Nprt) System.out.print(F[i] +spc2);//max limit
30          if(i>0 && i<=Nprt && i%line==0) System.out.println();//15 per line
31          }//end for
32       System.out.println("\n-----");
33    }//end Gen
```

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

Factorials Results

Lab 5

Java Long 17

```
Input N:
17
Factorials as gen:
1  2  6  24  120  720  5040  40320  362880  3628800  39916800  479001600
6227020800  87178291200  1307674368000  20922789888000
------
Factorials from array: 17
1  1  2  6  24  120  720  5040  40320  362880  3628800  39916800  479001600
6227020800  87178291200  1307674368000  20922789888000
--**--
```

```
Factorials by Jeff D
1 2 6 24 120 720 5040 40320 362880 3628800 39916800 479001600 1932053504 1278945280 2004310016 2004189184 -288522240
-- program is finished running --
```

MIPS assy

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	1952670022	1634300527	1646293868	1699356793	1142974054	1850277898	544503152	1769108595
0x10010020	3827566	1970302537	1919230068	561147762	0	1	2	6
0x10010040	24	120	720	5040	40320	362880	3628800	39916800
0x10010060	479001600	1932053504	1278945280	2004310016	2004189184	-288522240	0	0

Factorials Results

Lab 5

Java Long 17

```
Input N:
17
Factorials as gen:
1  2  6  24  120  720  5040  40320  362880  3628800  39916800  479001600
6227020800  87178291200  1307674368000  20922789888000
-----
Factorials from array: 17
1  1  2  6  24  120  720  5040  40320  362880  3628800  39916800  479001600
6227020800  87178291200  1307674368000  20922789888000
--**--
```

int 20

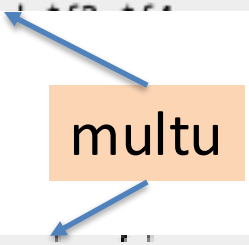
```
Input N:
20
Factorials as gen:
1  2  6  24  120  720  5040  40320  362880  3628800  39916800  479001600
1932053504  1278945280  2004310016  2004189184  -288522240  -898433024  109641728
```

12

```
Factorials by Jeff D
1 2 6 24 120 720 5040 40320 362880 3628800 39916800 479001600
-- program is finished running --
```

MIPS Multiply


```
mul $t1,$t2,$t3          Multiplication without overflow  : Set HI to high-order 32 bits, LO a
mul.d $f2,$f4,$f6        Floating point multiplication double precision : Set $f2 to double-pr
mul.s $f0,$f1,$f3        Floating point multiplication single precision : Set $f0 to single-pr
mult $t1,$t2             Multiplication : Set hi to high-order 32 bits, lo to low-order 32 bit
multu $t1,$t2            Multiplication unsigned : Set HI to high-order 32 bits, LO to low-orc
```




```
mfhi $t1                 Move from HI register : Set $t1 to contents of HI (
mflo $t1                 Move from LO register : Set $t1 to contents of LO (
```

MIPS MULT

Multiply

```
mult rs, rt
```

0	rs	rt	0	0x18
6	5	5	10	6

Unsigned multiply

```
multu rs, rt
```

0	rs	rt	0	0x19
6	5	5	10	6

Multiply registers `rs` and `rt`. Leave the low-order word of the product in register `lo` and the high-order word in register `hi`.

Multiply (without overflow)

```
mul rd, rs, rt
```

0x1c	rs	rt	rd	0	2
6	5	5	5	5	6

Put the low-order 32 bit of the product of `rs` and `rt` into register `rd`.

Multiply (with overflow)

```
mulo rdest, rsrc1, src2
```

pseudoinstruction

Unsigned multiply (with overflow)

```
mulou rdest, rsrc1, src2
```

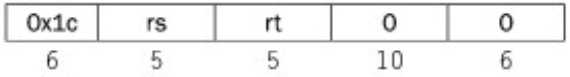
pseudoinstruction

Put the low-order 32 bits of the product of register `rsrc1` and `src2` into register `rdest`.

MIPS MULT

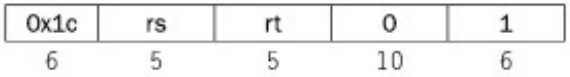
Multiply add

`madd rs, rt`



Unsigned multiply add

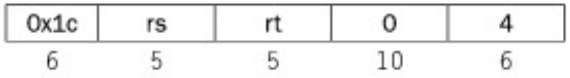
`maddu rs, rt`



Multiply registers `rs` and `rt` and add the resulting 64-bit product to the 64-bit value in the concatenated registers `lo` and `hi`.

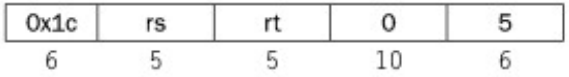
Multiply subtract

`msub rs, rt`



Unsigned multiply subtract

`msub rs, rt`



Multiply registers `rs` and `rt` and subtract the resulting 64-bit product from the 64bit value in the concatenated registers `lo` and `hi`.

MIPS MULT

- mulo (with overflow) – pseudo
- **rem** – remainder (modulo residue %)

Extended (pseudo) Instructions

```
rem $t1,$t2,$t3          REMainder : Set $t1 to (remainder of $t2 divided by $t3)
rem $t1,$t2,-100         REMainder : Set $t1 to (remainder of $t2 divided by 16-bit immediate)
rem $t1,$t2,100000       REMainder : Set $t1 to (remainder of $t2 divided by 32-bit immediate)
remu $t1,$t2,$t3         REMainder : Set $t1 to (remainder of $t2 divided by $t3, unsigned division)
```

- **div** – remainder in **$hi** (N mod *divisor*)

```
div $t1,$t2              Division with overflow : Divide $t1 by $t2 then set LO to quotient and HI to remainder (
```

Lab 5 Data


```
1  ## Lab 5 -- Facts
2  ## by Jeff Drobman
3  ##version: 1.0 >2-19-24
4  #register map
5  #$s0= size
6  #$t0=array ptr , #t1=count, $t2=F , $t3=i++
7  .eqv size, 20 #variable init.eqv size, 28 #variable init
8  .eqv npl, 8 #no. per line
9
10 .data
11 title: .asciiz  "Factorial sequence by Jeff D\n"
12 newLn: .asciiz "\n"
13 .align 2
14 space: .asciiz  "\t" #2 spaces
15 .align 2
16 begArray: .ascii ">GEB"
17 facts: .word  0:size #array allocation
18 .align 2
19 endArray: .ascii ">DNE" #END>
```

Lab 5 Setup


```
52  .text
53  GUI_msg(title) #popup title
54  #set up pointers
55  la $t0, facts #ptr
56  li $s0, size #final
57  subu $t1,$s0,1 #counter
58  #init 1st 2 numbers (1, 1)
59  li $t2,1
60  li $t3,2 #factor
61  sw $t2,($t0)
62  addi $t0,$t0,4 #incr ptr
```

Main Loop

Lab 5

```
63  loop_main:
64  multu $t2,$t3 #next fact
65  mflo $t2
66  #check hi for ovfl
67  jal checkHi
68  sw $t2,($t0)
69  addi $t0,$t0,4 #incr ptr
70  addiu $t3,$t3,1 #incr factor
71  subi $t1,$t1,1 #counter
72  bgtz $t1,loop_main
73  #call sub for print
74  jal print
75  done #macro for exit
76  #---end of main---
77  #subroutine:
78  checkHi:
79  #if hi<>0 then signal overflow
80  jr $ra
```

Use same **Print** sub as Lab 4

Binary Multiplication

Unsigned Multiplication

1 bit at a time

Y	Multiple	Op
0	0	none
1	1	add

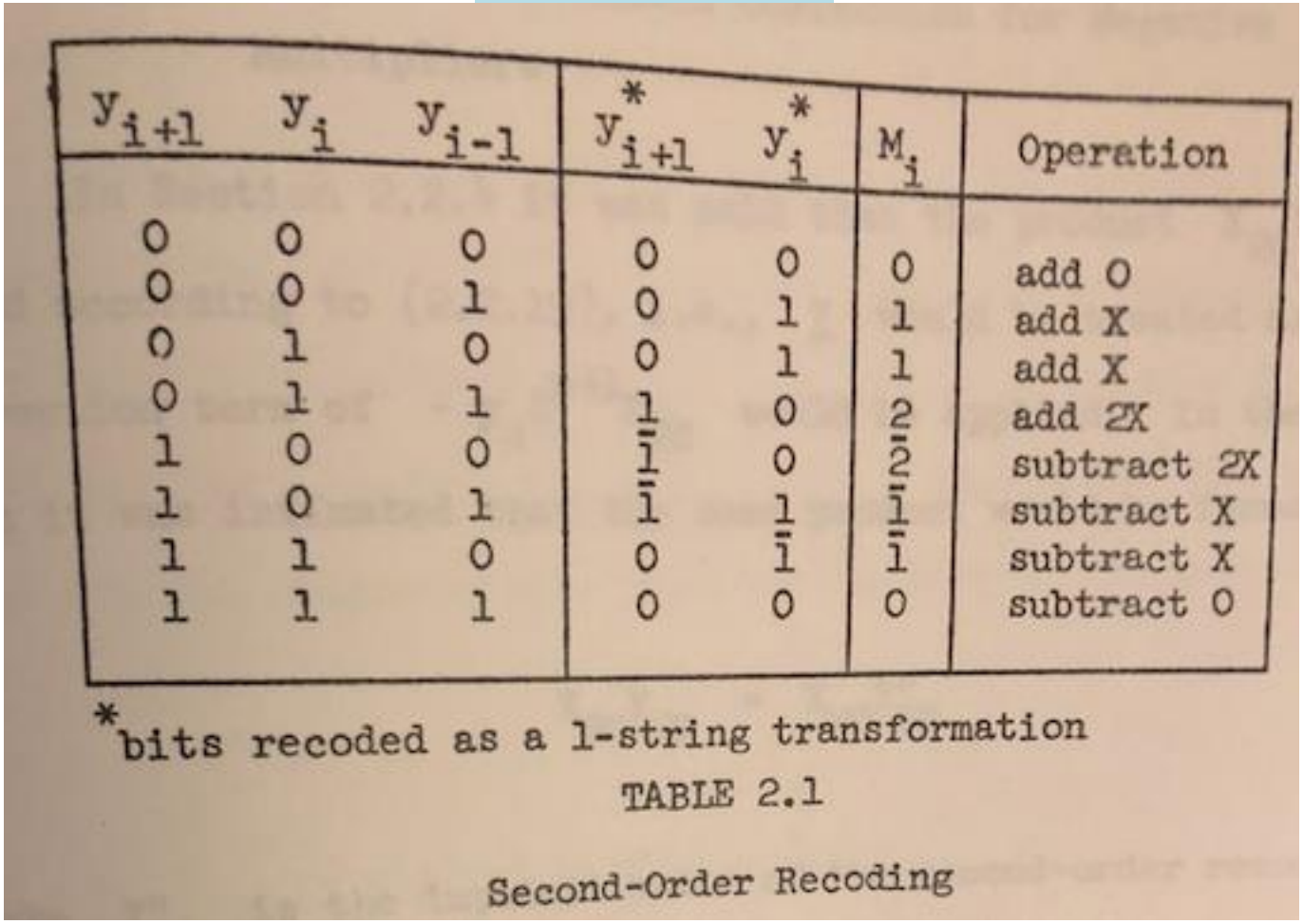
2 bits at a time

Y	Multiple	Op
00	0	none
01	1	add
10	2	Shift-add
11	3	Add twice

Binary Multiplication

Signed 2'sC Multiplication | Drobman MS Thesis

Booth's Recoding



y_{i+1}	y_i	y_{i-1}	y^*_{i+1}	y^*_i	M_i	Operation
0	0	0	0	0	0	add 0
0	0	1	0	1	1	add X
0	1	0	0	1	1	add X
0	1	1	1	0	2	add 2X
1	0	0	$\bar{1}$	0	$\bar{2}$	subtract 2X
1	0	1	$\bar{1}$	1	$\bar{1}$	subtract X
1	1	0	0	$\bar{1}$	$\bar{1}$	subtract X
1	1	1	0	0	0	subtract 0

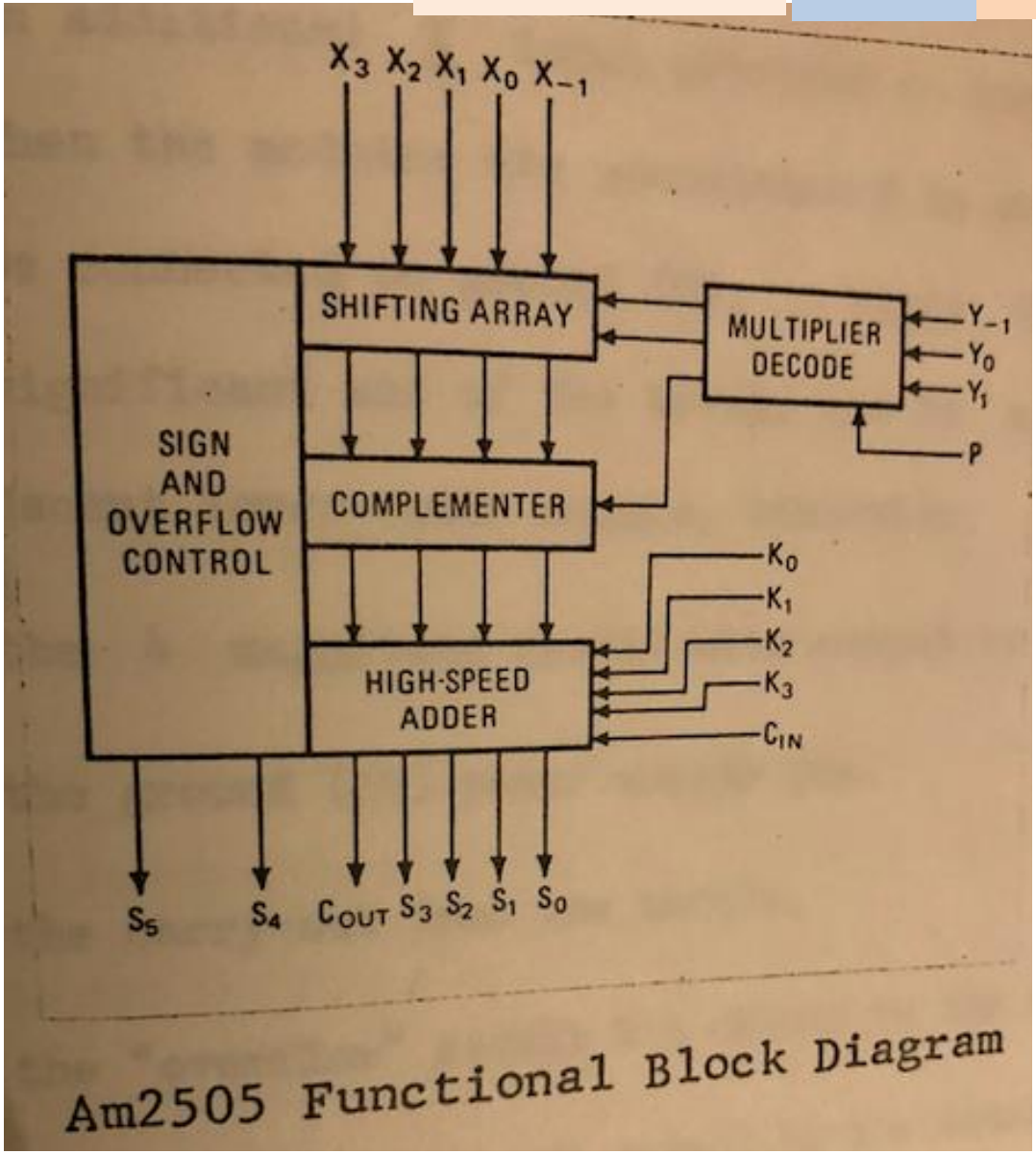
*bits recoded as a 1-string transformation

TABLE 2.1

Second-Order Recoding

Am2505 Multiplier

Drobman MS Thesis | Bit-slice | 1971-80



Am2505 Functional Block Diagram

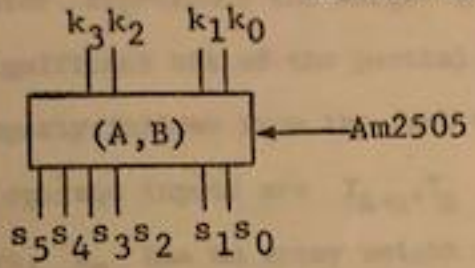
Am2505 Multiplier

Notation: (from [1], [15], [16])

multiplier pair = Y_{A+1}, Y_A

multiplicand group = $X_{B+3}, X_{B+2}, X_{B+1}, X_B$

"module designator" = (A,B)



2x4-bit slices

8-bit x 8-bit multiply

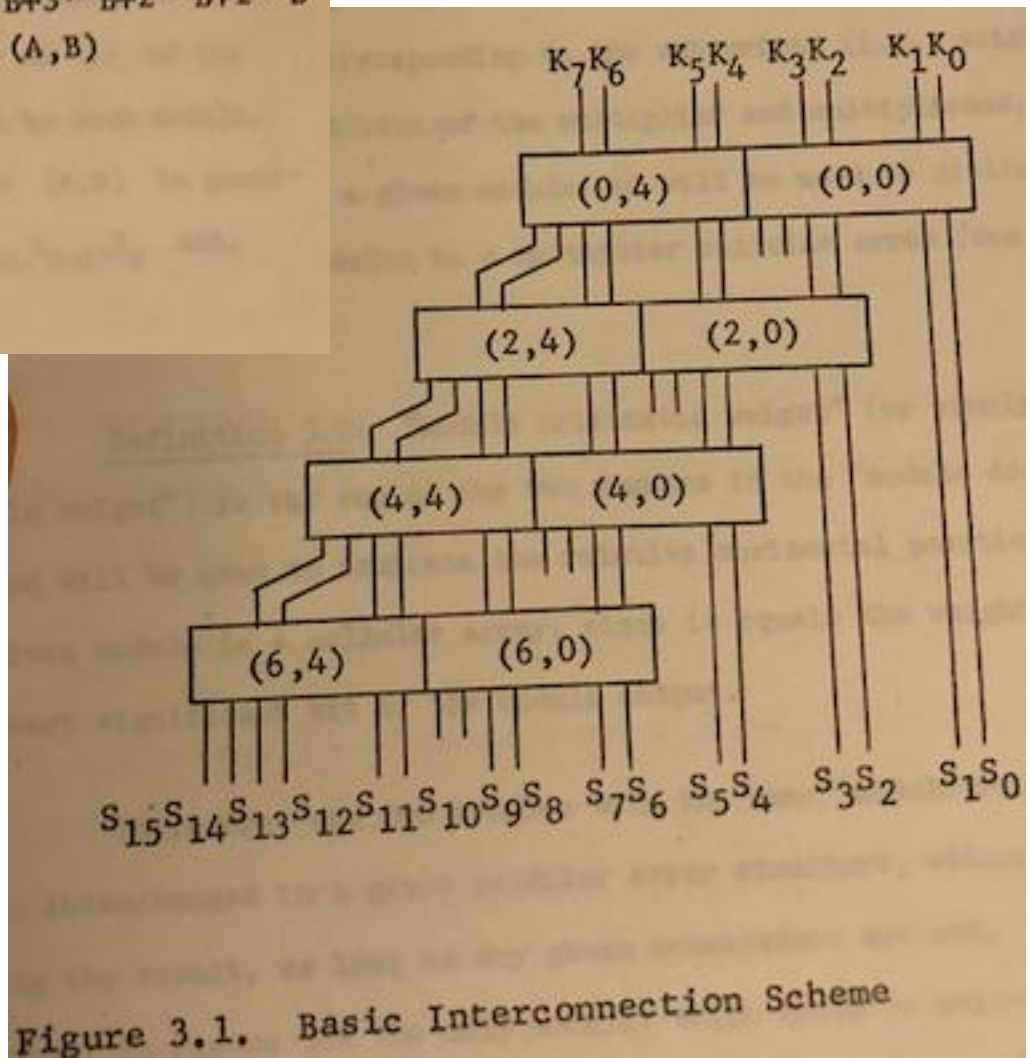
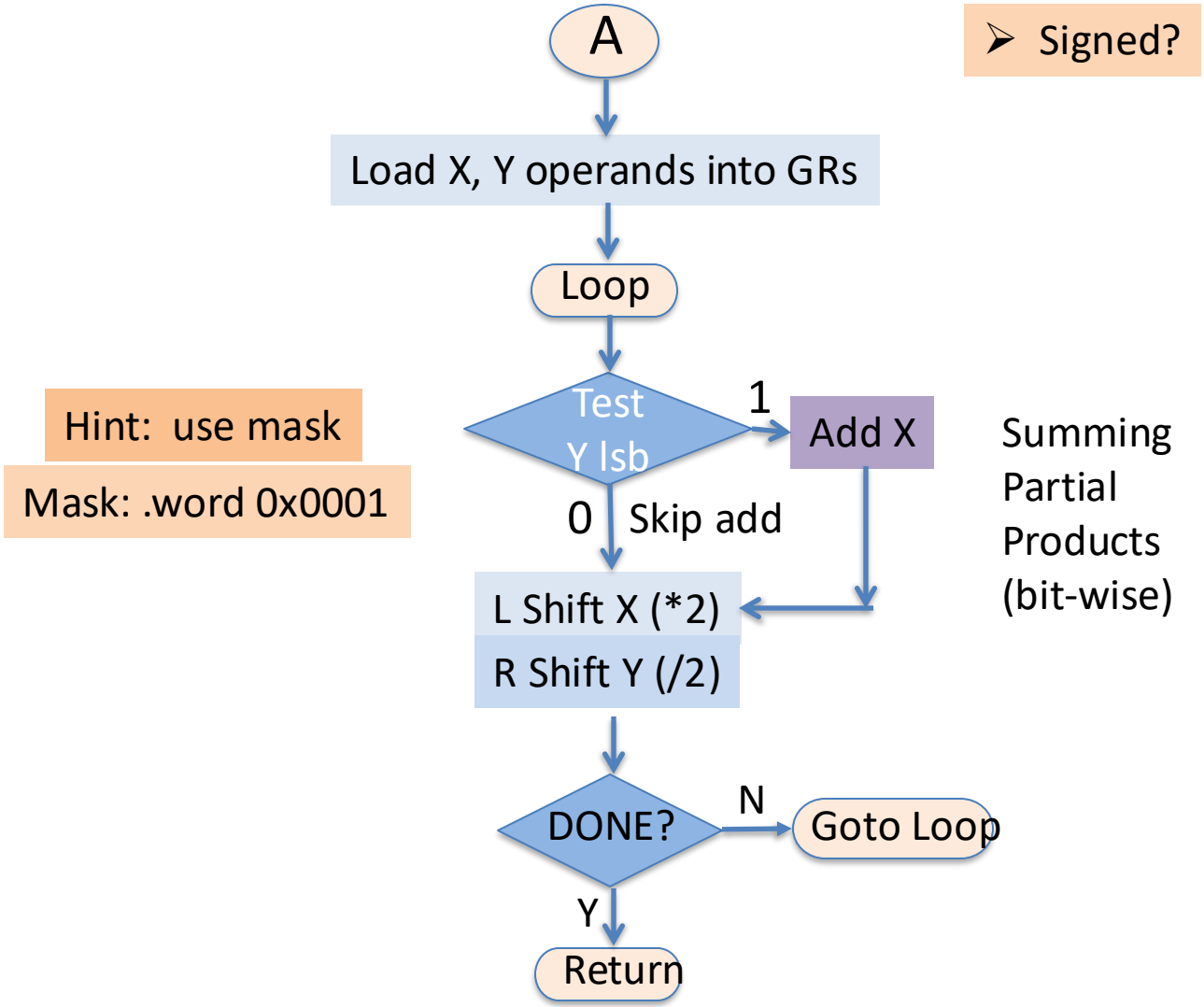


Figure 3.1. Basic Interconnection Scheme

Mult Subroutine



Factorials – Recursive

Lab 5

C or Java: *recursive*

```
//call Recursive method
System.out.println("Factorials-recursive: ");
int x = factRec(Ngen);
printit(Nprt,FR);
```

```
int factorial(int n) {
  if (n == 0)
    return 1;
  else
    return n * factorial(n-1);
}
```

```
51    //Recursive method
52    static int factRec(int N) {
53       System.out.print("FR:" +N +spc);
54       if(N==0) {
55          System.out.println();
56          return 1;}
57       //call recursively
58       FR[N] = N * factRec(N-1);//store
59       return FR[N];
60    }//end factRec
```

Store in array

Factorials – Recursive

Java: *recursive*

Lab 5

```
Input N:
17
Factorials as gen:
1   2   6   24   120   720   5040   40320   362880   3628800   39916800   479001600
1932053504   1278945280   2004310016   2004189184
-----
Factorials from array: 17
1   2   6   24   120   720   5040   40320   362880   3628800   39916800   479001600
1932053504   1278945280   2004310016   2004189184
------
Factorials-recursive:
FR:17 FR:16 FR:15 FR:14 FR:13 FR:12 FR:11 FR:10 FR:9 FR:8 FR:7 FR:6 FR:5 FR:4 FR:3
Factorials from array: 17
1   2   6   24   120   720   5040   40320   362880   3628800   39916800   479001600
1932053504   1278945280   2004310016   2004189184
```

2B

```
Factorials from array: 18
1   2   6   24   120   720   5040   40320   362880   3628800   39916800   479001600
1932053504   1278945280   2004310016   2004189184   -288522240
```

>>2B