Computer Architecture

MIPS Programming Model

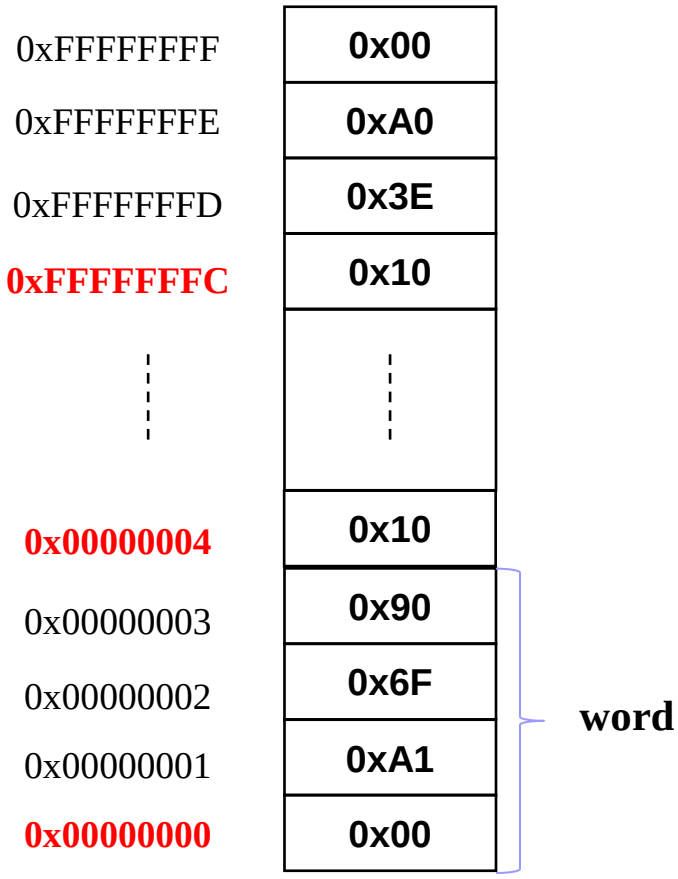
Hoon Oh
University Of Ulsan

MIPS Memory Layout

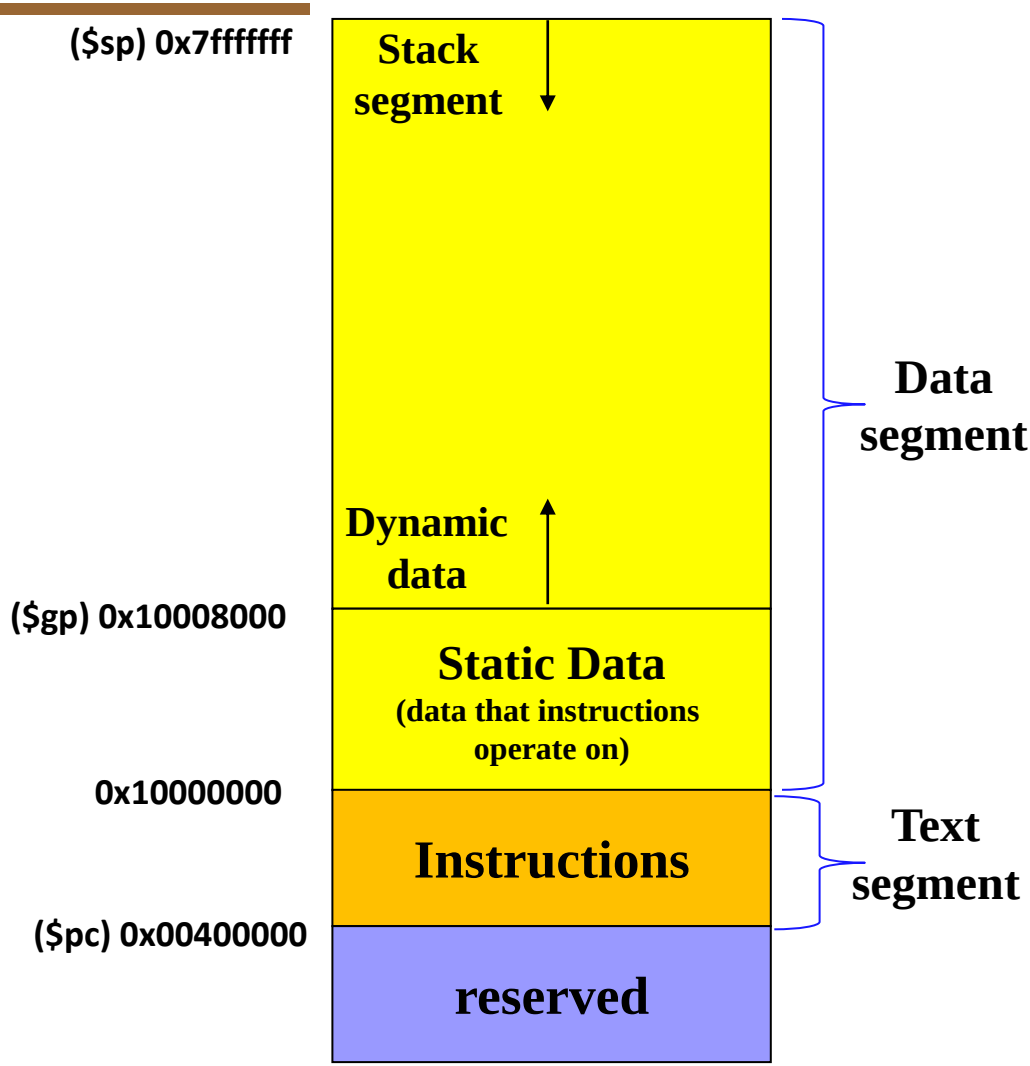
- Memory organization
  - Memory consists of a number of cells, each of which holding one byte.
  - Memory cells are numbered sequentially from zero up to the maximum allowable size (4GB) (assuming that the machine uses 32-bit address)

- Address space
  - Text segment
    - holds instructions for a program
  - Data segment
    - Static data
    - Dynamic data
      - Grows upward after static data.
  - Stack
    - Resides in the top of the address space and grows downward.

MIPS Memory Layout



Memory Organization



MIPS Memory Layout

MIPS registers

- MIPS processor contains 32 general purpose registers (numbered 0 – 31).
- Register *n* is designated by $n or R*n* (ex, R0 = $0).

- $zero (0) is always set to 0.
- $at (1), $k0 (26) and $k1 (27) are reserved for use by the assembler and OS.
- $v0 and $v1 (2, 3) are used for return values.
- $t0 - $t9 (8-15, 24, 25) are used for temporary values.
- $s0 - $s7(16-23) are used for long-lived values.
- $gp, $sp, $fp, $ra are used for special purposes, as shown in the table of the next page.

MIPS Registers

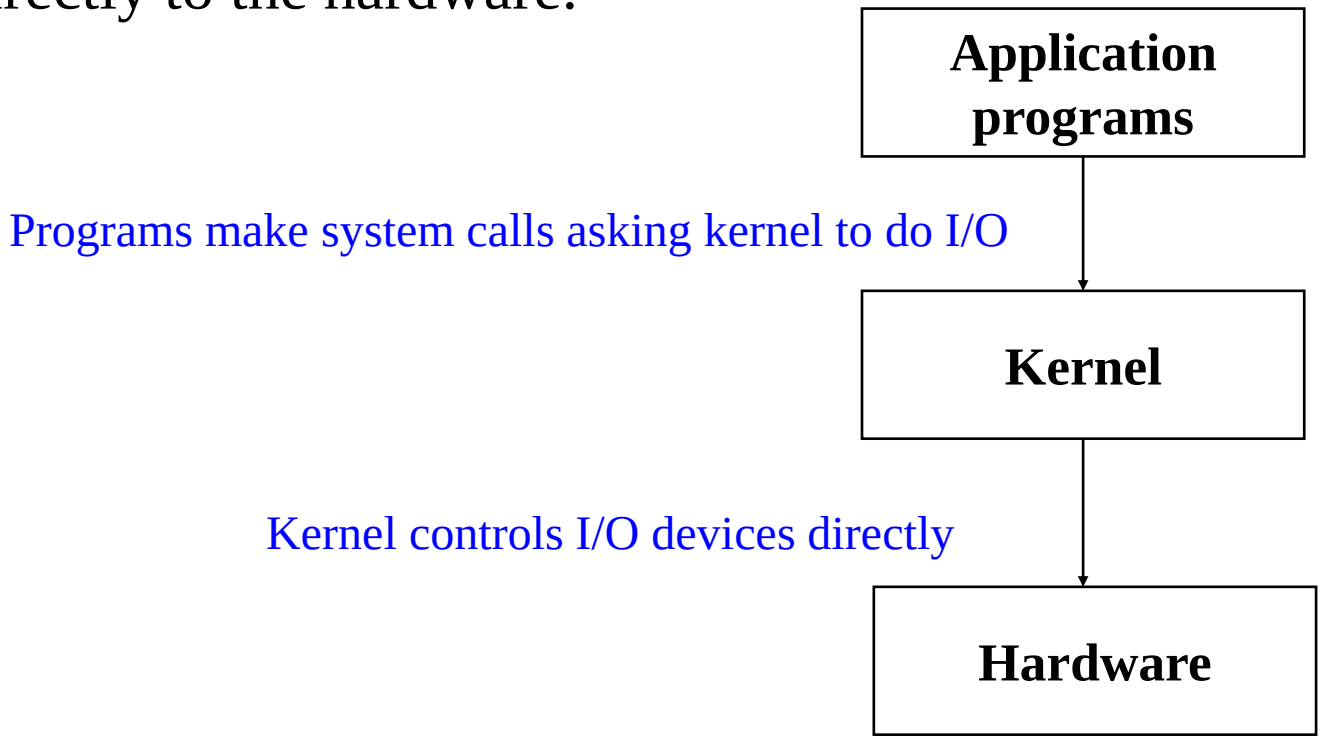
Register name	Number	Usage
zero	0	constant 0
at	1	reserved for assembler
v0, v1	2 ~ 3	expression evaluation and results of a function
a0 ~ a3	4 ~ 7	arguments 1 - 4
t0 ~ t7	8 ~ 15	temporary (not preserved across call)
s0 ~ s7	16 ~ 23	saved (preserved across call)
t8, t9	24, 25	temporary (not preserved across call)
k0, k1	26, 27	reserved for OS kernel
gp	28	pointer for global area
sp	29	stack pointer
fp	30	frame pointer
ra	31	return address (used by function call)

SPIM Simulator

- SPIM
  - Simulator that runs programs written for MIPS R2000/R3000 processors
- Advantages to using a machine simulator
  - MIPS workstation are not generally available.
  - Provides better environment for low-level programming than an actual machine:
    - detect more errors
    - provide more features
    - give convenient edit-assemble-load development cycle, compared to using a circuit board.
    - easy to identify and fix the bugs.
- Disadvantages
  - The programs run slower than on a real machine.

OS system calls

- An application program asks the kernel to do I/O by making system calls.
- The kernel implements these system calls by talking directly to the hardware.



SPIM's system calls

- SPIM provides a small set of 10 OS-like system services through the system call (syscall) instruction.

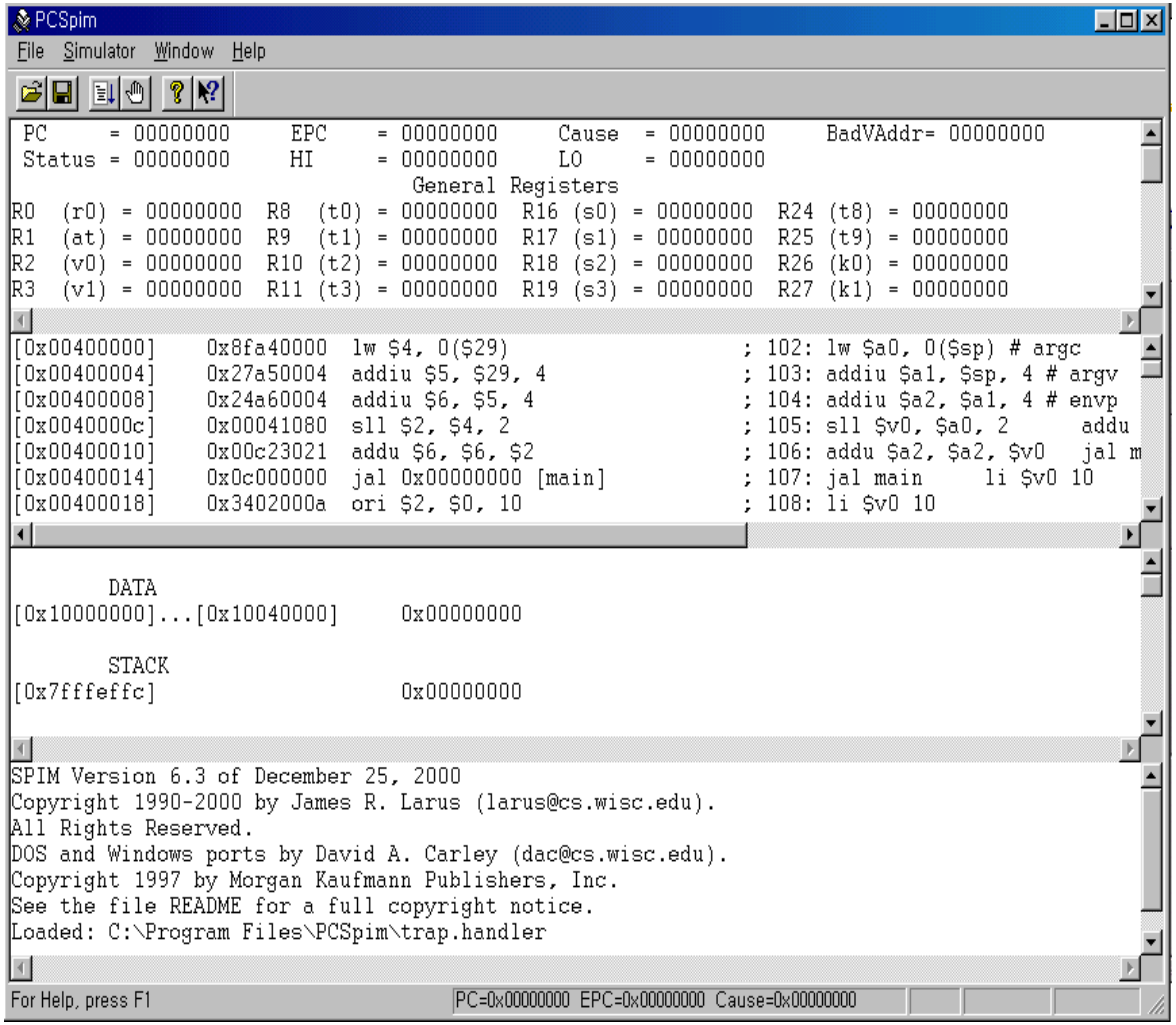
Service	Call code	Arguments	Results
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		

- $v0 : system call code
- $a0...$a3 : arguments
  ($f12 <- floating point values)
- syscall (initiate system service)
- $v0 :results
  ($f0: for the floating point)

- sbrk returns the address to a block of memory containing *n* additional bytes. Used for dynamic memory allocation.

PCSpim(starting)

- Registers window
- Text segment
- Data Segment
- Messages window



PCSpim(4 Windows)

- Registers window
  - shows the values of all registers in the MIPS CPU and FPU
- Text segment

```
[0x00400000]  0x8fa40000  lw  $4,  0($29)  ;   102  :  lw  $a0,  0($sp)
```

  - address of instruction (hexadecimal)
  - instruction's numerical encoding (hexadecimal)
  - instruction's mnemonic description
  - line number in assembly file
  - actual line of assembly file
- Data Segment
  - data loaded into program's memory
  - data on the program's stack
- Messages window
  - messages for SPIM(including error messages)

PCSpim(Loading, Running, Debugging)

- Loading
  - File -> Open
  - select assembly file
- Running
  - Simulator -> Go
  - results are displayed in console
- Single step
  - Simulator -> Single Step(F10)
  - run an instruction at a time
- Multiple steps
  - Simulator -> Multiple Step…(F11)
  - run given number of instructions at a time
- Breakpoint
  - Simulator -> Breakpoints(Ctrl + B)
  - stop program immediately before it executes a particular instruction

Code Format

- Source code format

  [label:] operation [operand], [operand], [operand] [#comment]

  - Brackets ([ ]) indicates an optional field.

- Constants
  - Hex numbers: prefix with 0x
  - String: “hello world\n”
  - Special characters: **\n** (newline), **\t** (tab), **\”** (quote).
  - SPIM directives for character strings

    .ascii “abcd” : do not null-terminate them.

a	**b**	**c**	**d**	?	?

    .asciiz “abcd” : null-terminate them.

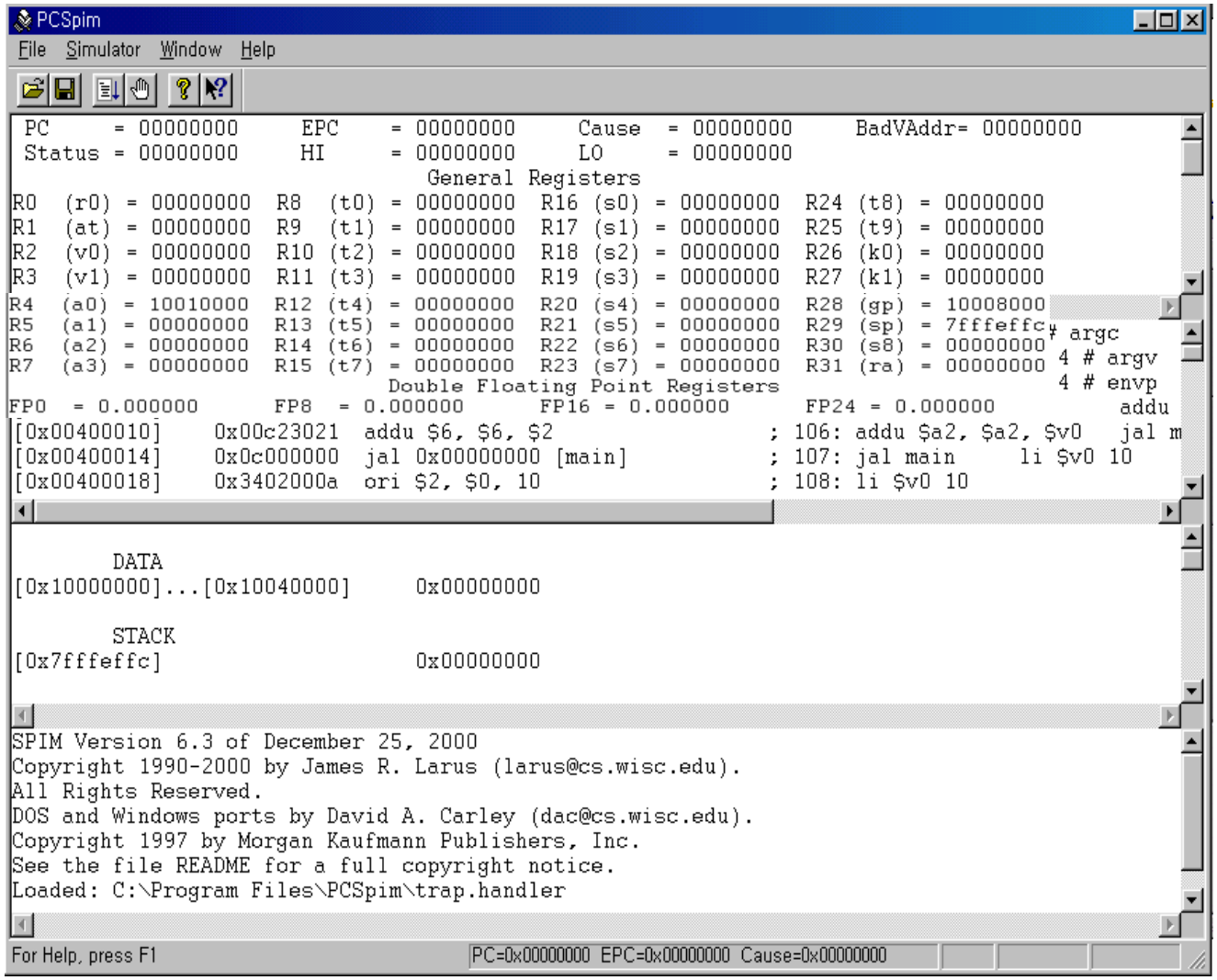
a	**b**	**c**	**d**	**\0**	?

Code Format: hello.a

```
1   ## hello.a - prints out "hello world"
2   ##          a0 - points to the string
3   ##################################################################
4   #                          text segment                          #
5   ##################################################################
6           .text
7           .globl  __start
8   __start:                        # execution starts here
9           la $a0, str             # put string address into a0
10          li $v0, 4               # system call to print
11          syscall                 # out a string
12
13          li $v0, 10
14          syscall                 # au revoir (goodbye) ...
15  ##################################################################
16  #                          data segment                          #
17  ##################################################################
18          .data
19  str:    .asciiz "hello world\n"
20  ##
21  ## end of file hello.a
```

PC-SPIM for Windows

- **Registers window**
- **Text segment**
- **Data Segment**
- **Messages window**



Inside text & data segments

$at = $1 : reserved fro assembler

```
[0x00400000] 0x3c011001 lui $1, 4097 [str] ;9: la $a0, #str put string ..
[0x00400004] 0x34200000 ori $4, $1, 0 [str]
[0x00400008] 0x34020004 ori $2, $0, 4       ;10: li $v0, #system call to ..
[0x0040000c] 0x0000000c syscall             ;11; syscall
[0x00400010] 0x3402000a ori $2, $0, 10      ;13; li $v0, 10
[0x00400014] 0x0000000c syscall             ;14; syscall
```

text segment

0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

= 4097

```
DATA
[0x10000000] ... [0x000fffx] 0x00000000
[0x1000fffc] 0x00000000
[0x10010000] 0x6c6c6568 0x6f77206f 0x0a646c72 0x00000000
[0x10010010] ... [0x10020000] 0x00000000
```

lleh ow o \ndlr 000\0

```
STACK
[0x7fffaffc] 0x00000000
```

```
        .data
str:    .asciiz "hello world\n"
```

data segment

MIPS 산술연산 예

	Example	Comments
Add	add $t1, $t2, $t3	# $t1 = $t2 + $t3
Add immediate	addi $t1, $t2, 50	# $t1 = $t2 + 50
Subtract	sub $t1, $t2, $t3	# $t1 = $t2 - $t3
Multiply	mult $t2, $t3	# Hi, Lo = $t2 x $t3
Divide	div $t2, $t3	# Lo = $t2 / $t3 # Hi = $t2 mod $t3
Move from Hi	mfhi $t1	# $t1 = Hi # get copy of Hi
Move from Li	mflo $t1	# $t1 = Lo # get copy of LoInstruction

Programming Example: temp.a

```
1.  ## temp.a ask user for temperature in Celsius.
2.  ##   convert to Fahrenheit, print the result. (F = 9C/5 + 32)
3.  ##            v0 - reads in Celsius
4.  ##            t0 - holds Fahrenheit result
5.  ##            a0 - points to output strings
6.  ###############################################################
7.  #                        text segment                         #
8.  ###############################################################
9.        .text
10.       .globl __start
11. __start:
12.       la $a0, prompt    # print prompt on terminal
13.       li $v0,4
14.       syscall
15.       li $v0,5          # syscall 5 reads an integer
16.       syscall
17.       mul $t0,$v0,9     # to convert,multiply by 9,
18.       div $t0,$t0,5     # divide by 5, then
19.       add $t0,$t0,32    # add 32
```

temp.a (cont...)

```
20.       la   $a0,ans1      # print string before result
21.       li   $v0,4
22.       syscall
23.
24.       move $a0,$t0       # print result
25.       li   $v0,4
26.       syscall
27.
28.       move $a0,endl      # system call to print
29.       li   $v0,4         # out a newline
30.       syscall
31.
32.       li   $v0,10
33.       syscall  # au revoir ...
34.
35. ############################################################
36. #                   data segment                           #
37. ############################################################
38.
39.       .data
40. prompt: .asciiz  "Enter temperature (Celsius): "
41. ans1:   .asciiz  "The temperature in Fahrenheit is "
42. end1:   .asciiz  "\n"
43. ## end of file temp.a
```

math1.a

```
1.  ## Question:  calculate A * X^2 + B * X + C
2.  ## Output format: "answer =180"
3.  ################################################################
4.  #                     text segment                             #
5.  ################################################################
6.        .text
7.        .globl __start
8.  __start:                # execution starts here
9.  # Any changes outside of two dashed lines will be discarded
10. # by mipsmark. put your answer between dashed lines.
11. # ----------------- start cut ------------------------
12.                        (해답)
13. # ------------------ end cut --------------------------
14. ################################################################
15. #                     data segment                             #
16. ################################################################
17.       .data
18. X:    .word 7
19. A:    .word 3
20. B:    .word 4
21. C:    .word 5
22. ans:  .asciiz "answer = "
23. endl: .asciiz "\n"
24. ## End of file math1.a
```

Solution: math1.a

```
1.  # ------------------ start cut -------------------------
2.  # This solution has a bug. Perform source level debugging to
3.  # single step through the code and locate error (RED lines move?).
4.      lw $t0, X
5.      lw $t1, A
6.      lw $t2, B
7.      lw $t3, C
8.      mul $t4, $t0, $t0          # t4 = X^2
9.      mul $t4, $t4, $t1          # t4 = A * X^2
10.     mul $t5, $2, $t0           # t5 = B * X
11.     add $t4, $t4, $t5          # t4 = A * X^2 + B * X
12.     add $t4, $t4, $t3          # t4 = A * X^2 + B * X + C
13.     la $a0, ans                # system call to print
14.     li $v0, 4                  # out string
15.     syscall
16.     move $a0, $t4              # print result on terminal
17.     li $v0, 1
18.     syscall
19.     move $a0, endl             # system call to print
20.     li $v0, 4                  # out a newline
21.     syscall
22.     li $v0, 10
23.     syscall                    # au revoir ...
24. # ------------------ end cut -------------------------
```

STUDY-AID1

- There exists one bug in math1.a. Using step button, check the contents of the following registers after executing the fifth line (**`lw $t1, A`**).
- PC, t0, t1, gp, sp

- Check the following registers before and after executing the 10th line (**`mul $t5, $2, $t0)`**.
  - PC
  - t4, t5
  - Check whether or not the content of t5 B*X=4 x 7 = 28 (0x1C) after execution?

- Check if the value is 180 after modifying the error.

STUDY-AID2

- **Write and run the assembly program that computes 5 * X^2 – 3 (use hw1-math2.a).**
  Assign X to 7
  Output: Answer = 242

STUDY-AID3

- **Write and run the assembly program that compute (NUM-3)*(NUM+4) (use hw2-math3.a).**

  Assign NUM to 10

  Output: Answer = 98

STUDY-AID4

- **Write and run the recursive program for Fibonacci function (use hw3-fib-recursive.a)**

STUDY-AID5

- What is the relationship between machine language and Assembly?
- What are the advantages of Assembly programming compared to C programming?
- Show the procedure regarding how the computer perform 5 – 3.
  - Note that computer can perform ADD only.
- What is the role of program counter?
- What is the rule in making an identifier in SPIM?
- What are the difference between syntax error and logical error?
- What is the purpose of using single stepping in SPIM?
- What is the breakpoint in SPIM?
- Whar are the objectives of hi and lo registers?
- State the differences of la, lb, li, and lw.