SPIM 시뮬레이터

SPIM이란

MIPS-32 프로세서의 어셈블리(assembly) 프로그램을 실행할 수 있는 소프트웨어 시뮬레이터

최신버젼 QtSpim_9.1.19_Windows.msi (https://sourceforge.net/projects/spimsimulator/files/)을 설치

설치 디렉토리 `C:\Program Files (x86)\QtSpim`

QtSpim.exe : MSVCP110.dll이 없다는 오류가 발생 할 경우 MS의 공식사이트(https://www.microsoft.com/ko-kr/download/confirmation.aspx?id=30679)에서 vcredist_x86 (32-bit), vcredist_x64 (64-bit) 두 프로그램을 다운로드 받아서 모두 설치

QtSpim 사용법

QtSpim 매뉴얼 https://www.lri.fr/~de/QtSpim-Tutorial.pdf 참조

- 텍스트에디터(notepad, atom등)로 example.asm 또는 example.s와 같은 형식으로 어셈블리 파일을 작성
- QtSpim을 수행해서, `load` 버튼을 이용해서 example.s 파일을 읽어들이고, `run` 버튼을 이용해서 프로그램 실행

PCSpim-Cache

캐시 시뮬레이션이 가능하도록 확장한 시뮬레이터로, 사이트 http://www.disca.upv.es/spetit/spim.htm 소스와 수행파일 다운로드

- 파일 수행시 exception 파일 디렉토리가 달라서 오류가 발생하면, exception 파일을 연결하겠다고 하고 소스파일의 압축을 풀어서 spim-cache-080605/spim-cache/src/exceptions.s 연결

MIPS Instruction Set

partial list

In all examples, $1, $2, $3 represent registers. For class, you should use the register names, not the corresponding register numbers.

Arithmetic Instructions

Instruction	Example	Meaning	Comments
add	add $1,$2,$3	$1=$2+$3	
subtract	sub $1,$2,$3	$1=$2-$3	
add immediate	addi $1,$2,100	$1=$2+100	"Immediate" means a constant number
add unsigned	addu $1,$2,$3	$1=$2+$3	Values are treated as unsigned integers, not two's complement integers
subtract unsigned	subu $1,$2,$3	$1=$2-$3	Values are treated as unsigned integers, not two's complement integers

Arithmetic Instructions

Instruction	Example	Meaning	Comments
add immediate unsigned	addiu $1,$2,100	$1=$2+100	Values are treated as unsigned integers, not two's complement integers
Multiply (without overflow)	mul $1,$2,$3	$1=$2*$3	Result is only 32 bits!
Multiply	mult $2,$3	$hi,$low=$2*$3	Upper 32 bits stored in special register `hi` Lower 32 bits stored in special register `lo`
Divide	div $2,$3	$hi,$low=$2/$3	Remainder stored in special register `hi` Quotient stored in special register `lo`
Unsigned Divide	divu $2,$3	$hi,$low=$2/$3	$2 and $3 store unsigned values. Remainder stored in special register `hi` Quotient stored in special register `lo`

Logical

Instruction	Example	Meaning	Comments
and	and $1,$2,$3	$1=$2&$3	Bitwise AND
or	or $1,$2,$3	$1=$2	$3
and immediate	andi $1,$2,100	$1=$2&100	Bitwise AND with immediate value
or immediate	or $1,$2,100	$1=$2	100
shift left logical	sll $1,$2,10	$1=$2<<10	Shift left by constant number of bits
shift right logical	srl $1,$2,10	$1=$2>>10	Shift right by constant number of bits

Data Transfer

Instruction	Example	Meaning	Comments
load word	lw $1,100($2)	$1=Memory[$2+100]	Copy from memory to register
store word	sw $1,100($2)	Memory[$2+100]=$1	Copy from register to memory
load upper immediate	lui $1,100	$1=100x2^16	Load constant into upper 16 bits. Lower 16 bits are set to zero.
load address	la $1,label	$1=Address of label	Pseudo-instruction (provided by assembler, not processor!) Loads computed address of label (not its contents) into register
load immediate	li $1,100	$1=100	Pseudo-instruction (provided by assembler, not processor!) Loads immediate value into register

Data Transfer

Instruction	Example	Meaning	Comments
move from hi	mfhi $2	$2=hi	Copy from special register hi to general register
move from lo	mflo $2	$2=lo	Copy from special register lo to general register
move	move $1,$2	$1=$2	Pseudo-instruction (provided by assembler, not processor!) Copy from register to register.

Variations on load and store also exist for smaller data sizes:

- 16-bit halfword: `lh` and `sh`
- 8-bit byte: `lb` and `sb`

Conditional Branch

All conditional branch instructions compare the values in two registers together.
If the comparison test is true, the branch is taken (i.e. the processor jumps to the new location).
Otherwise, the processor continues on to the next instruction.

Instruction	Example	Meaning	Comments
branch on equal	beq $1,$2,100	if($1==$2) go to PC+4+100	Test if registers are equal
branch on not equal	bne $1,$2,100	if($1!=$2) go to PC+4+100	Test if registers are not equal
branch on greater than	bgt $1,$2,100	if($1>$2) go to PC+4+100	Pseduo-instruction
branch on greater than or equal	bge $1,$2,100	if($1>=$2) go to PC+4+100	Pseduo-instruction

Conditional Branch

Instruction	Example	Meaning	Comments
branch on less than	blt $1,$2,100	if($1<$2) go to PC+4+100	Pseduo-instruction
branch on less than or equal	ble $1,$2,100	if($1<=$2) go to PC+4+100	Pseduo-instruction

Note 1: It is much easier to use a label for the branch instructions instead of an absolute number. For example: beq $t0, $t1, equal. The label "equal" should be defined somewhere else in the code.

Note 2: There are many variations of the above instructions that will simplify writing programs! Consult the Resources for further instructions, particularly Textbook Appendix A.

Comparison

Instruction	Example	Meaning	Comments
set on less than	slt $1,$2,$3	if($2<$3)$1=1; else $1=0	Test if less than. If true, set $1 to 1. Otherwise, set $1 to 0.
set on less than immediate	slti $1,$2,100	if($2<100)$1=1; else $1=0	Test if less than. If true, set $1 to 1. Otherwise, set $1 to 0.

Note: There are many variations of the above instructions that will simplify writing programs! Consult the Resources for further instructions, particularly Textbook Appendix A.

Unconditional Jump

Instruction	Example	Meaning	Comments
jump	j 1000	go to address 1000	Jump to target address
jump register	jr $1	go to address stored in $1	For switch, procedure return
jump and link	jal 1000	$ra=PC+4; go to address 1000	Use when making procedure call. This saves the return address in $ra

Note: It is much easier to use a label for the jump instructions instead of an absolute number. For example: j loop. That label should be defined somewhere else in the code.

System Calls

- The SPIM simulator provides a number of useful system calls.
    - These are *simulated*, and *do not represent MIPS processor instructions*. (They would be implemented by the operating system and/or standard library in computer.)
- System calls are used for input and output, and to exit the program.
    - They are initiated by the `syscall` instruction.
    - You must first supply the appropriate arguments in registers $v0, $a0-$a1, or $f12, depending on the specific call desired. (In other words, not all registers are used by all system calls).
    - The syscall will return the result value (if any) in register $v0 (integers) or $f0 (floating-point).

System Calls

Available syscall services in SPIM:

Service	Operation	Code (in $v0)	Arguments	Results
print_int	Print integer number (32 bit)	1	$a0 = integer to be printed	None
print_float	Print floating-point number (32 bit)	2	$f12 = float to be printed	None
print_double	Print floating-point number (64 bit)	3	$f12 = double to be printed	None
print_string	Print null-terminated character string	4	$a0 = address of string in memory	None
read_int	Read integer number from user	5	None	Integer returned in $v0
read_float	Read floating-point number from user	6	None	Float returned in $f0

System Calls

Service	Operation	Code (in $v0)	Arguments	Results
read_double	Read double floating-point number from user	7	None	Double returned in $f0
read_string	Works the same as Standard C Library fgets() function.	8	$a0 = memory address of string input buffer $a1 = length of string buffer (n)	None
sbrk	Returns the address to a block of memory containing n additional bytes. (Useful for dynamic memory allocation)	9	$a0 = amount	address in $v0
exit	Stop program from running	10	None	None

System Calls

Service	Operation	Code (in $v0)	Arguments	Results
print_char	Print character	11	$a0 = character to be printed	None
read_char	Read character from user	12	None	Char returned in $v0
exit2	Stops program from running and returns an integer	17	$a0 = result (integer number)	None

Notes:

- The `print_string` service expects the address to start a null-terminated character string. The directive `.asciiz` creates a null-terminated character string.

System Calls

Notes:

- The `read_int`, `read_float` and `read_double` services read an entire line of input up to and including the newline character.
- The `read_string` service has the same semantics as the C Standard Library routine fgets().
    - The programmer must first allocate a buffer to receive the string
    - The `read_string` service reads up to n-1 characters into a buffer and terminates the string with a null character.
    - If fewer than *n-1* characters are in the current line, the service reads up to and including the newline and terminates the string with a null character.
- There are a few additional system calls not shown above for file I/O: **open, read, write, close** (with codes 13-16)

Assembler Directives

An assembler directive allows you to request the assembler to do something when converting your source code to binary code.

Directive	Result
.word w1, ..., wn	Store n 32-bit values in successive memory words
.half h1, ..., hn	Store n 16-bit values in successive memory words
.byte b1, ..., bn	Store n 8-bit values in successive memory words
.ascii str	Store the ASCII string str in memory. Strings are in double-quotes, i.e. "Computer Science"
.asciiz str	Store the ASCII string str in memory and **null-terminate** it. Strings are in double-quotes, i.e. "Computer Science"
.space n	Leave an empty n-byte region of memory for later use
.align n	Align the next datum on a 2^n byte boundary. For example, .align 2 aligns the next value on a word boundary

Registers

32 general-purpose registers have been divided into groups and used for different purposes.

- Registers have both
  a **number** (used by the hardware) and
  a **name** (used by the assembly programmer).

Register Number	Register Name	Description	Register Number	Register Name	Description
$0	$zero	The value 0	$1	$at	Assembler temporary
$2-$3	$v0-$v1	(values) from expression evaluation and function results	$4-$7	$a0-$a3	(arguments) First four parameters for subroutine
$8-$15	$t0-$t7	Temporary variables	$16-$23	$s0-$s7	Saved values representing final computed results
$24-$25	$t8-$t9	Temporary variables	$26-$27	$k0-$k1	Reserved for OS Kernel
$28	$gp	Global pointer	$29	$sp	Stack pointer
$30	$fp	Frame pointer	$31	$ra	Return address

Bubble sort (C code)

```
void swap(int v[], int k)
{
        int temp;

        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
}

void sort (int v[], int n)
{
        int i, j;
        for (i = 0; i < n; i += 1) {
                for (j = i - 1;
                     j >= 0 && v[j] > v[j + 1];
                     j -= 1) {
                        swap(v,j);
                }
        }
}
```

Bubble sort (Assembly code)

```
swap:   sll $t1, $a1, 2          # $t1 = k * 4
        add $t1, $a0, $t1        # $t1 = v+(k*4)
                                 # (address of v[k])
        lw $t0, 0($t1)           # $t0 (temp) = v[k]
        lw $t2, 4($t1)           # $t2 = v[k+1]
        sw $t2, 0($t1)           # v[k] = $t2 (v[k+1])
        sw $t0, 4($t1)           # v[k+1] = $t0 (temp)
        jr $ra                   # return to calling routine
```

Bubble sort (Assembly code)

```
sort:   addi $sp,$sp, -20       # make room on stack for 5 registers
        sw $ra, 16($sp)         # save $ra on stack
        sw $s3,12($sp)          # save $s3 on stack
        sw $s2, 8($sp)          # save $s2 on stack
        sw $s1, 4($sp)          # save $s1 on stack
        sw $s0, 0($sp)          # save $s0 on stack

        move $s2, $a0           # save $a0 into $s2
        move $s3, $a1           # save $a1 into $s3
        move $s0, $zero         # i = 0
for1tst: slt $t0, $s0, $s3      # $t0 = 0 if $s0 ≥ $s3 (i ≥ n)
        beq $t0, $zero, exit1   # go to exit1 if $s0 ≥ $s3 (i ≥ n)
        addi $s1, $s0, -1       # j = i - 1
for2tst: slti $t0, $s1, 0       # $t0 = 1 if $s1 < 0 (j < 0)
        bne $t0, $zero, exit2   # go to exit2 if $s1 < 0 (j < 0)
        sll $t1, $s1, 2         # $t1 = j * 4
        add $t2, $s2, $t1       # $t2 = v + (j * 4)
        lw $t3, 0($t2)          # $t3 = v[j]
        lw $t4, 4($t2)          # $t4 = v[j + 1]
        slt $t0, $t4, $t3       # $t0 = 0 if $t4 ≥ $t3
        beq $t0, $zero, exit2   # go to exit2 if $t4 ≥ $t3
        move $a0, $s2           # 1st param of swap is v (old $a0)
        move $a1, $s1           # 2nd param of swap is j
        jal swap                # call swap procedure
        addi $s1, $s1, -1       # j -= 1
        j for2tst               # jump to test of inner loop
exit2: addi $s0, $s0, 1         # i += 1
        j for1tst               # jump to test of outer loop

exit1: lw $s0, 0($sp)           # restore $s0 from stack
        lw $s1, 4($sp)          # restore $s1 from stack
        lw $s2, 8($sp)          # restore $s2 from stack
        lw $s3,12($sp)          # restore $s3 from stack
        lw $ra,16($sp)          # restore $ra from stack
        addi $sp,$sp, 20        # restore stack pointer
        jr $ra
```

A demonstration of some simple MIPS instructions used to test QtSPIM

```
        # Declare main as a global function
        .globl main

        # All program code is placed after the
        # .text assembler directive
        .text

# The label 'main' represents the starting point
main:
        li $t2, 25              # Load immediate value (25)
        lw $t3, value           # Load the word stored in value (see bottom)
        add $t4, $t2, $t3       # Add
        sub $t5, $t2, $t3       # Subtract
        sw $t5, Z               #Store the answer in Z (declared at the bottom)

        # Exit the program by means of a syscall.
        # There are many syscalls - pick the desired one
        # by placing its code in $v0. The code for exit is "10"
        li $v0, 10      # Sets $v0 to "10" to select exit syscall
        syscall         # Exit

        # All memory structures are placed after the
        # .data assembler directive
        .data

        # The .word assembler directive reserves space
        # in memory for a single 4-byte word (or multiple 4-byte words)
        # and assigns that memory location an initial value
        # (or a comma separated list of initial values)
value:  .word 12
Z:      .word 0
```

참고사이트

https://www.joinc.co.kr/w/Site/Assembly/Documents/Spim/spim-chapter9

https://ecs-network.serv.pacific.edu/ecpe-170/tutorials

숙제

n! (팩토리얼)을 구하는 MIPS 어셈블리 프로그램을 작성하고, 수행결과를 QtSpim을 이용해서 확인해보세요.

1. 반복을 사용해서 구하는 프로그램을 작성하시오.
   $n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$
2. 리커젼(Recursion; 재귀순환)을 이용한 어셈블리 프로그램을 작성하시오
   $n! = n \times (n-1)!$

각자 수행하고, 1, 2번에 대한 소스코드와 QtSpim을 이용한 수행결과를 제출