A Web-Based Graphical Education Support Tool for Teaching the Course of Assembly Language Programming

URL: http://cis.k.hosei.ac.jp/~yamin/asm/

Yamin Li, Department of Computer Science, Hosei University, yamin@hosei.ac.jp
Assembly Language Programming

- The foundation of many machines, from general-purpose computer systems to embedded systems, lies in assembly language programming and computer architecture.
- If someone doesn’t understand assembly language, he (she) can neither develop a compiler nor design a CPU.
- The assembly language can be used to demonstrate what is actually happening inside the computer.
- It is an important path to understanding how the computer works at the machine level.
Some special (hardware) operations may not be controlled in C or Java

Example: How to select one of the following round-modes of FPU in C?

- Round to nearest (to even if g r s = 1 0 0)
- Round toward minus infinity (−∞)
- Round toward plus infinity (+∞)
- Round toward zero (also called truncate)

Some assembly codes must be inserted in C

```
define Near asm volatile("fldcw _RoundNear")
int _RoundNear = 0x103f; // Round mode = 00
Near; // Round to nearest```
Top Three Assembly Programming Languages

- High-level languages are machine independent
- Assembly languages are machine dependent
- Top three assembly programming languages
  - Intel x86
  - ARM
  - MIPS

- We selected MIPS instruction set
  - MIPS is a typical RISC instruction set
  - To design MIPS CPUs in the following course of Computer Organization and Design
The basic assembly programs’ assembling and simulation:

- Grammar checking
- Translating to binary code
- Single-step execution
- Break-point setting
- Showing the contents of the register file
- Showing the contents of the data memory

The basic input and output system function calls, like `scanf()` and `printf()` in C, or `readLine()` and `System.out.print()` in Java
Two Existing MIPS Simulators

- **SPIM** — Windows (PCSpim) and Linux (spim and xspim)
  - Register display that shows the values of registers
  - Control buttons that let us interact with simulator
  - Text segment that display instructions
  - Data and stack segment
  - Spim messages used to write messages

- **MARS** — Java-based simulator
  - Provided a self-contained program editor
  - Showed the displacements with labels
  - Changed the simulator layout
AsmSim — A MIPS Simulator We Developed

- Web-based — does not need installation
- Graphical — supports VGA
- The AsmSim provides three windows:
  - A main window that provides control buttons and displays the instructions, the values of registers, and the data of the memory
  - A graphics console window that is used to input command, to display messages, and to show the images of Video RAM (VRAM)
  - An online editor window for editing the assembly program
/*
 * hello.s
 * prints "Hello, World!" to console window
 * input: $4: the string address
 */

.text
main:
    subi $sp, $sp, 24
    sw $ra, 20($sp)
    sw $fp, 16($sp)
    move $fp, $sp

user_main:
    la $4, msg
    jal printf

return_to_caller:
    move $sp, $fp
    lw $fp, 16($sp)
    lw $ra, 20($sp)
    addu $sp, $sp, 24
    jr $ra

.data
msg:
    .asci "Hello, World!\n"
.end
The Functions of AsmSim

- The basic assembling and simulation
- The basic input and output system function calls
- Interrupt mechanism for handling the keyboard interrupt
- Manipulating a standard VGA (640 × 480 pixels)
- Graphics object draw and fill
- Generating Xilinx COE and Altera MIF for FPGAs
- Some other useful function calls, like `get a random number`, `get the timer`, `sleep`, and `get calendar`
- Supporting the data structure of the linked-list for implementing some graph algorithms
The Main Window of AsmSim

![AsmSim Main Window](image_url)
The Highlighted Program Edit Window of AsmSim

```
/*
 * template s, for editing your codes
 * Control_s for searching (quit search with mouse or arrow key pressing)
 * Copy & Paste: See http://cis.h-esei.ac.jp/ yamin/asm/CopyAndPaste.html
 */
.text
main:
    # code segment
    # program entry
    subu $sp, $sp, 64
    # reserve stack space
    sw $ra, 60($sp)
    # save return address, important
    sw $fp, 56($sp)
    # save frame pointer
    move $fp, $sp
    # new frame pointer
user_main:
    # insert your code here (replace "nep")
return_to_caller(os):
    move $sp, $fp
    # restore stack pointer
    lw $fp, 56($sp)
    # restore frame pointer
    lw $ra, 60($sp)
    # restore return address, important
    addu $sp, $sp, 64
    # release stack space
    jr $ra
    # return to operating system
.data
a_string:
    # data segment
    # string address
    .asciiz "Welcome.\n"
    # a string, not used
a_word:
    # word address
    .word 0xffffffff
    # an integer (-1), not used
.end
```
The Graphics Console Window of AsmSim

VGA (640 x 480)
11 Buttons in the Main Window

1. **(edit)**: opens the program edit window
2. **(step)**: executes one instruction that is highlighted
3. **(goto)**: executes instructions to the break-point
4. **(ascii)**: displays the ascii of selected data
5. **(restart)**: re-loads the user program
6. **(run)**: executes user program
7. **(stop)**: stops the execution
8. **(quit)**: quits and enters *command line mode*
9. **(inst)**: displays the encodes of the MIPS instructions
10. **(xilinx)**: generates Xilinx COE file
11. **(altera)**: generates Altera MIF file
## Instructions AsmSim Supported

![Processor Instructions](image)

### MIPS32 Instruction Format (32 bits)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs, rt, rd</td>
<td>Source registers</td>
</tr>
<tr>
<td>imm</td>
<td>Immediate value</td>
</tr>
<tr>
<td>add, sub, and, ori, slt</td>
<td>Instruction type</td>
</tr>
<tr>
<td>rs, rt, rd</td>
<td>Target register</td>
</tr>
</tbody>
</table>

### Examples

- **add**: Source registers 
- **sub**: Source registers
- **and**: Source registers
- **ori**: Source registers
- **slt**: Source registers

**Notes**:
- Instructions are designed for compatibility with MIPS32 architecture.
- Immediate values are supported for specific operations.
- Special instructions like j, jal, jr, lw, sw, b, beq, bne, li, beqz, bnez are also supported for control flow and data manipulation.
1. `getchar()` gets a character from STDIN
2. `putchar()` puts a character to STDOUT
3. `scanf()` reads an input from STDIN
4. `printf()` writes output(s) to STDOUT
5. `sprintf()` writes output(s) to a buffer
6. `malloc()` allocates a block of memory
7. `gettimer()` gets the system timer in ms
8. `getrandom()` gets a random integer number
9. `getarrow()` gets an arrow key’s information
10. `getcal()` gets the calendar in integers
11. `getcals()` gets the calendar in string
12. `sleep()` suspends execution for an interval of time
13. `paint()` shows the image on the VGA
System Calls AsmSim Supported

14. `drawline()` draws a color line
15. `drawoval()` draws a color oval
16. `drawrect()` draws a color rectangle
17. `drawrect3d()` draws a color 3D rectangle
18. `drawstring()` draws specified text
19. `filloval()` fills a color oval
20. `fillrect()` fills a color rectangle
21. `fillrect3d()` fills a color 3D rectangle
22. `refresh_vga_manu()` refreshes VGA by `paint()`
23. `refresh_vga_auto()` refreshes VGA automatically
24. `key_event_ena()` enables keyboard interrupt
25. `key_event_dis()` disables keyboard interrupt
Interrupt Supported in AsmSim

Main program

Interrupt handler

Keyboard interrupt

__Key_Event:

eret

Before transferring to __Key_Event, EPC ← return address
eret (return from exception): PC ← EPC
The Graphics Output Example (graphics.s)
The Graphics Output Example (vram.s)
The Graphics Output Example (key_event.s)
; Do not use the syscall functions provided in this simulator.
; single memory module:
memory_initialization_radix=2;
memory_initialization_vector=
0010000T110111101110110000000000111000,
101011110111111000000000111100,
101011111011111000000000111100,
00000000001111011110000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000,
% Do not use the syscall functions provided in this simulator.
% single memory module:
DEPTH = 32; % Memory depth and width are required
WIDTH = 32; % Enter a decimal number
ADDRESS_RADIX = HEX; % Address and value radixes are optional
DATA_RADIX = BIN; % Enter BIN, DEC, HEX, or OCT; unless
% otherwise specified, radixes = HEX

CONTENT
BEGIN
0000 : 001000111011110111111111111111110000000;
0001 : 1010111101111111000000000000111100;
0002 : 1010111101111111000000000000111100;
0003 : 000000000000111101111111000000000000000;
0004 : 000000000000000000000000000000000000000;
0005 : 000000000000000000000000000000000000000;
0006 : 1000111101111111000000000000111100;
0007 : 1000111101111111000000000000111100;
0008 : 0010001101111111000000000000111100;
0009 : 0000001111000000000000000000000000;
000a : 0000000000000000000000000000000000;

Support for Linked-List Data Structure

'define NULL 0x0
.data
v1: .word 1, p1
p1: .word v1, p2
p2: .word v2, p3
p3: .word v4, NULL
v2: .word 2, p4
p4: .word v5, p5
p5: .word v3, NULL
v3: .word 3, p6
p6: .word v1, p7
p7: .word v2, p8
p8: .word v5, NULL
v4: .word 4, p9
p9: .word v5, p10
p10: .word v3, NULL
v5: .word 5, p11
p11: .word v4, NULL
Depth First Search (DFS):

forall (x in V) visited[x] = false;
push(s);
while (!stack_empty()) {
    x = pop();
    if (!visited[x]) {
        visited[x] = true;
        forall (w in adj[x])
            if (!visited[w]) push(w);
    }
}

debug> quit
[asmain]$ asmain dfs.s
debug> run
debug> v1
debug> v4
debug> v3
debug> v5
debug> v2
debug> The program finished
debug>
Breadth First Search (BFS):

forall (x in V) visited[x] = false;
cnqncuc(s);
while (!queue_empty()) {
    x = dequeue();
    if (!visited[x]) {
        visited[x] = true;
        forall (w in adj[x])
            if (!visited[w]) enqueue(w);
    }
}
To Let Copy & Paste Work

- Find “java.policy” file. It is usually located at 
  C:\Program Files\Java\jre6\lib\security\ 
- Edit it: Add the following 3 lines in the top of the file:

  ```
  grant codeBase "http://cis.k.hosei.ac.jp/-" {
  permission java.awt.AWTPermission "accessClipboard";
  };
  ```

- Save it as “.java.policy” at your home directory (fold) 
  (Note that this filename starts with a dot)

- Close the Web Browser and open it again

- You should have Copy & Paste enabled now
The assembly language programming is an important way to understanding how the computer works at the machine level.

AsmSim supports:
- System calls like `printf()` and `scanf()` in C
- Graphics draws and fills
- VRAM and VGA accesses
- Keyboard interrupt mechanism
- Linked list data structure
- Automatic generation of the memory initialization files for Xilinx and Altera FPGAs

AsmSim can be used by anyone anywhere anytime.
THX