CSCE 312 LAB 5

By Dr. Eun Jung Kim

Announcements

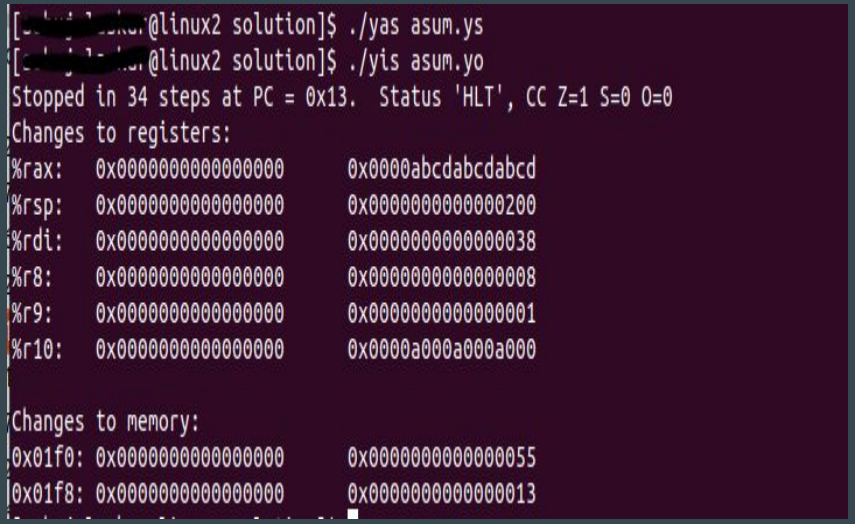
- Deadline: Mar 31, 11:59 PM
- Individual Lab
- There will be no DEMO, grading will be based on the report.

SETUP for LAB5

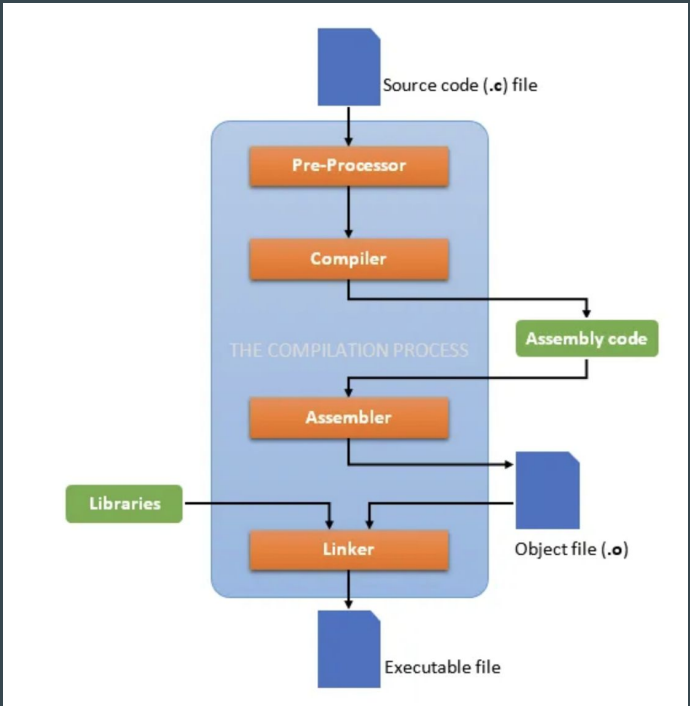
- To setup your environment, use the **linux.cse.tamu.edu** server
- Download the simulator zip file from this link.
- Binaries required for this lab are already available in “binary(only_for_linux_cse_server)” folder.
- However, we only tested these binaries in linux.cse.tamu.edu server.

SETUP for LAB5

- To test your binaries are working properly, go to ‘y86-code’ folder and run following commands(Make sure both binaries are in ‘y86-code’ folder as well):
  - ./yas asum.ys
  - ./yis asum.yo
- The first command converts the Y86-64 assembly code (.ys) into object machine code (.yo)
- The 2nd command simulates the instructions and prints out the changes in the state of registers and memory.



Steps in Program Execution



- Pre processing: gets rid of comments, replaces macros, includes header files
- Compiler: Parses through the code and converts the high level code to an Intermediate Representation [Assembly Code]
- Assembler: Takes this Assembly code and converts it into binary
- Linker: links multiple such binaries so that functions from other binaries can be used and finally an executable file is created

Y86-64 [Assembly Code]

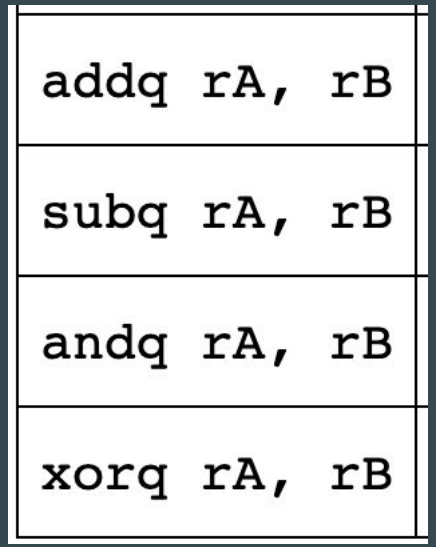
- Instruction Set Architecture similar and much simpler form of x86
- PROGRAM COUNTER – indicates address of current instruction that is being implemented
- REGISTERS - 64bit meaning each register is of length 64bits
- MEMORY (DRAM) – Each address corresponds to 64bit
- CONDITION FLAGS – 1 bit flags set by arithmetic and logical operations
  - OF : Overflow
  - ZF : Zero
  - SF : Negative
- STATUS REGISTER - indicates state of current execution
  - AOK : Normal Operation
  - HLT : Program halted
  - INS : Invalid instruction encountered
  - ADR : Bad address encountered

REGISTERS

- Each register has 4-bit ID

`%rax`	0	`%r8`	8
`%rcx`	1	`%r9`	9
`%rdx`	2	`%r10`	A
`%rbx`	3	`%r11`	B
`%rsp`	4	`%r12`	C
`%rbp`	5	`%r13`	D
`%rsi`	6	`%r14`	E
`%rdi`	7		F

Y86-64 Instruction Set - Operations



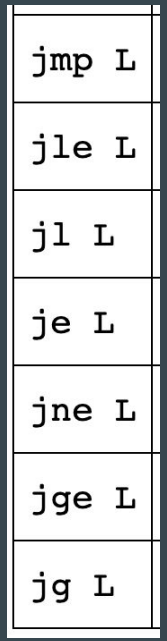
`Let rdx = 5, rax = 2`

- addq rA rB - add two register value (Reg[rB] = Reg[rB]+Reg[rA])
  - addq %rdx, %rax
  - result : rax : 7, rdx : 5
- subq rA rB - subtract two register value (Reg[rB] = Reg[rB]-Reg[rA])
  - subq %rax, %rdx
  - result : rdx : 3, rax : 2

`Let rdx = 6(110), rax = 4(100)`

- andq rA rB - bitwise and operation, Reg[rB] = Reg[rB] & Reg[rA]
  - andq %rdx, %rax
  - Result : rax : 4, rdx: 6
- xorq rA rB: bitwise exclusive-or operation, Reg[rB] = Reg[rB] ^ Reg[rA]
  - xorq % rdx, %rax
  - result : rax : 2, rdx : 6

Y86-64 Instruction Set - Jump



- jmp - jump to that destination
- jle - if less or equal , jump to Dest if last result ≤ 0
- jl - if less, jump to Dest if last result < 0
- je - if equal, jump to Dest if last result = 0
- jne - if not equal, jump to Dest if last result ≠ 0
- jge - if greater or equal, jump to Dest if last result ≥ 0
- jg - if greater, jump to Dest if last result > 0

Y86-64 Mov Instructions

Instruction	Source	Destination
`rrmovq rA, rB`	Register	Register
`irmovq V, rB`	Immediate	Register
`rmmovq rA, D(rB)`	Register	Memory
`mrmovq D(rA), rB`	Memory	Register

- irmovq : immediate value (number) into register
  - irmovq $3, %rax , will store value 3 to the rax register
- rrmovq : move one register value to another
  - rrmovq %rax, %rcx , now rcx has value 3
- rmmovq : move register value to memory
  - rmmovq %rax, 0(%rbx) ; assume ebx : 0x4000, now at address 0x4000, it contains 3
  - rmmovq %rax, 4(%rbx) ; number in front : offset, add 4 to that memory address, 0x4004 contains 3
- mrmovq : move value from memory to a register
  - mrmovq 4(%rbx), %rdx; move value at 0x4004 into rdx, which is 3 now.

Other Instructions

We have other instructions as well such as:

- Push
- Pop
- Call
- Ret

More details:
https://csit.kutztown.edu/~schwesin/fall21/csc235/lectures/Instruction_Set_Architecture.pdf

PROBLEM 1, 2

- Write the correct y86-64 assembly code in .ys files for the C code given in the question
- Generate .yo files from .ys files:
  - > ./yas <filename>.ys
- Execute the code from the .yo file:
  - > ./yis <filename>.yo
- After successful execution it shows you the changes in registers and memory.

- Helpful Resource:
  - Bryant Book 3.1 - 3.6
  - Codes in y86-64 folder

```
int i,j;
…..
if (i < j) {
            i=i-3;
            j=j+4;
}
else {
            i=7;
            j++;
}
```

Problem 1

```
int j,k;
…..
for (int i=1; i <=10; i++) {
          j = i*2;
          k = j-4;
}
```

Problem 2

PROBLEM 3

- Generate x86 assembly code in .s files from the C code given to you and analyze the assembly code as instructed in the questions
  - > gcc –S <filename>.c –o <filename>.s
  - .s file has the assembly code

```
//Program 1, file name "lab5_prob3_1.c"
     #include <stdio.h>
     int main(int argc, char *argv[]){
          printf("Hello, world\n");
          return 0;
     }


//Program 2, file name "lab5_prob3_2.c"
     #include <stdio.h>
     int main(int argc, char *argv[]) {
          int i = 2;
          i++;
          printf("The value of  i is %d\n", i);
          return 0;
     }
```

- Helpful Resource:
  - https://medium.com/@laura.derohan/compiling-c-files-with-gcc-step-by-step-8e78318052

PROBLEM 4

- Generate x86 assembly code in .s files from the C code given to you and analyze the assembly code as instructed in the questions

```
//File 1, named "lab5_prob4_main.c"
     #include <stdio.h>
     void print_hello();
     int main(int argc, char *argv[])
     {
               print_hello();
               return 0;
     }


 void print_hello(){
          printf("Hello, world\n");
 };
```

PROBLEM 5

- Generate assembly and object files from the given C code. Then link those object files to get the executable. Finally, analyze the assembly code as instructed in the questions.

```
//File 1, named "lab5_prob5_main.c"
     void print_hello();
     int main(int argc, char *argv[]) {
          print_hello();
          return 0;
     }

//File 2, named "lab5_prob5_print.c"
     #include <stdio.h>
     void print_hello() {
               printf("Hello, world\n");
     };
```

PROBLEM 6

- Use inline x86 assembly to add assembly code to a C program.
- How to do that?
  - Resource_1
  - Resource_2
  - Resource_3
  - Resource_4

```
//File named "lab5_prob6.c"
     #include <stdio.h>
     int very_fast_function(int i){
               if ( (i*18 - 3) <= 300) return i++;
               else return 0;
     }

     int main(int argc, char *argv[]) {
               int i;
               i=16;
          printf("The function value of  i is %d\n",
          very_fast_function(i) );
               return 0;
     }
```

Files to be submitted as a zip file

- Report with all screenshots
- yas executable
- yis executable
- .yo and .ys files for Problem 1 and Problem 2
- .s files for Problem 3,4,5
- .c file for Problem 6