

Homework 2
695.744
T. McGuire
Johns Hopkins University

1. Throughout the semester, you will be utilizing tools such as **IDA Pro, objdump, Windbg, gdb** and other debugging utilities. One feature these tools share is the ability to turn *machine code* into human readable *assembly language*. It is important to have an understanding of how these tools work. In doing so, you will have a better idea of how to make the tool work better for you. The goal of this project is to create a program that can turn *machine code* into *assembly language*. You must support adding labels when jumps or calls are being used. Keep in mind that you may jump or call forwards! You will only be required to handle a small subset of the Intel Instruction Set (see below for the required mnemonics).

Assumptions that can be made:

- a) Code starts at offset 0 in the given file. This means you do not have to worry about the headers that certain linkers add.
- b) You only have to implement the given mnemonics.
- c) If you hit an unknown opcode, your program should handle this gracefully and give feedback to the user as to the cause of the problem. **Do not exit upon receiving an invalid opcode.**
- d) You must handle jumping/calling forwards and backwards (including using labels).
- e) This must work on the sample that is supplied, but it must also work on other tests as well.
- f) **Implement using the linear sweep algorithm. Be sure to explain the strengths and weaknesses of each algorithm discussed in class.**

Supported Mnemonics

***For all instructions below, do not worry about the ESP register being a destination register. ESP is sometimes handled differently and you are not expected to handle that.**

***All references will be 32-bit references. For example, you do not need to handle “mov dl, byte [ebx]”, you only need to handle “mov edx, dword [ebx]”.**

*** Your output must be similar to the examples below**

*** You must implement labels (as seen in the Example 2)**

| | |
|--------|--------|
| add | or |
| and | pop |
| bswap | popcnt |
| call | push |
| cmp | retn |
| dec | sal |
| idiv | sar |
| imul | sbb |
| inc | shl |
| jmp | shr |
| jz/jnz | test |
| lea | xor |
| mov | |
| mul | |
| neg | |
| nop | |
| not | |

For the 'shl'/'shr' instructions, you only need to support:

shr r/m32, 1

shl r/m32, 1

For the 'jz'/'jnz'/'jmp' you must implement:

jz 32-bit displacement

jz 8-bit displacement

jnz 32-bit displacement

jnz 8-bit displacement

jmp 32-bit displacement

jmp reg

jmp [reg]

jmp [reg + disp32]

For the 'retn' instruction family, you must implement:

retn

retn 16-bit value

For the 'mov' and similar instructions, you must implement:

mov reg1, reg2

mov [reg1], reg2

mov [reg1+ disp32], reg2

mov reg1, [reg2]

mov reg1, [reg2+ disp32]

mov reg1, [reg2+ disp8]

mov reg1, imm32

mov [reg1], imm32

mov [reg1+ disp32], imm32

mov [disp32], imm32

mov eax, [disp32]

Example 1: With no jumps

```
0:    31 c0                xor    eax, eax
2:    01 c8                add    eax, ecx
4:    01 d0                add    eax, edx
6:    55                  push   ebp
7:    89 e5                mov    ebp, esp
9:    52                  push   edx
a:    51                  push   ecx
b:    b8 44 43 42 41      mov    eax, 041424344h
10:   8b 55 08            mov    edx, [ebp + 08h]
13:   8b 4d 0c            mov    ecx, [ebp + 0ch]
16:    01 d1                add    ecx, edx
18:    89 c8                mov    eax, ecx
1a:    5a                  pop    edx
1b:    59                  pop    ecx
1c:    5d                  pop    ebp
1d:    c2 08 00            ret    08h
```

Example 2: With a conditional jump

```
0:    55                  push   ebp
1:    89 e5                mov    ebp, esp
3:    52                  push   edx
4:    51                  push   ecx
5:    39 d1                cmp    ecx, edx
7:    74 0f                jz     offset_18h
9:    b8 44 43 42 41      mov    eax, 041424344h
e:    8b 55 08            mov    edx, [ebp+08h]
11:   8b 4d 0c            mov    ecx, [ebp+0ch]
14:    01 d1                add    ecx, edx
16:    89 c8                mov    eax, ecx
offset_18h
18:    5a                  pop    edx
19:    59                  pop    ecx
1a:    5d                  pop    ebp
1b:    c2 08 00            ret    08h
```