Programming Assignment 1

695.744

T. McGuire
Johns Hopkins University

Introduction

Throughout the semester, you will be utilizing tools such as **Ghidra**, **IDA Pro**, **objdump**, **Windbg**, **gdb** and other debugging/disassembling 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*.

Deliverables

1. Brief discussion (about 1 page) on the strengths and weaknesses of the recursive descent and linear sweep algorithms. What makes tools like IDA and Ghidra powerful disassemblers?
2. Your source code for a disassembler for a small subset of the Intel Instruction Set, as described in the remainder of this assignment.

1 Requirements

Your disassembler must:

- Be written in any of the following programming languages: C, C++, Go, Rust, Java, Python. Please ask the instructor if you have issues with this requirement.
- Not crash on any (in)valid inputs.
- Use either the linear sweep or recursive descent algorithm. Most students choose linear sweep.
- Print disassembled instructions to standard output.
- Handle jumping/calling forwards and backwards, adding labels where appropriate with the following form (see Example 2 below).

  `offset_XXXXXXXXh:`
- Handle unknown opcodes by printing the address, the byte, and the assembly as follows (see skeleton code for an example).

  `db <byte>`
- Work on the supplied examples in addition to other test files that are not supplied.
- Implement only the given opcodes detailed in the Supported Mnemonics section.
- Implements both **`SIB`** and **`MODRM`** bytes.
- Have the input file specified using the "**`-i`**" command-line option.
- Display only addresses, instruction machine code (i.e. the bytes that make up the instruction), disassembled instructions/data, and labels.

2 Assumptions

The following assumptions may be made when writing your disassembler:

- The input file is a binary that contains some x86 machine code.
- Code starts at offset 0 in the given file. Do not worry about headers that are generally added by linkers.
- You will start the assignment the week it is assigned.
- If you have any questions, feel free to reach out to me! I'm always happy to help clarify and point you in the right direction.

3 Supported Mnemonics

For all instructions below, remember that addressing modes **0b00**, **0b01**, and **0b10** with an **r/m32** operand value of **0b100** indicates that a **SIB** follows the **MODRM** byte. In the Intel SDM this is noted by an Effective Address of **[--][--]**. Examples are given in the next section and you may reference the Intel Instruction Tutorial for more details.

All register 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]**". An immediate will be a 32-bit value while the displacement *may* be 8-bits or 32-bits in size. The only exceptions to this are the "**retn imm16**" and "**retf imm16**" instructions.

add	**jmp**	**pop**
and	**jz/jnz**	**push**
call	**lea**	**repne cmpsd**
clflush	**mov**	**retf**
cmp	**movsd**	**retn**
dec	**nop**	**sub**
idiv	**not**	**test**
inc	**or**	**xor**

4 Instruction Details

4.1 add/and/mov/not/or/pop/push/sub/etc.

For these and similar instructions, you must implement (where applicable):

```
add r/m32, imm32
add r32, imm32
add r32, [ disp32 ]
add r32, r/m32
add r32, [ r/m32 + disp8 ]
add r32, [ r/m32 + disp32 ]
add r32, [ r/m32*1 + disp32 ]
add r32, [ r/m32*2 + disp32 ]
add r32, [ r/m32*4 + disp32 ]
add r32, [ r/m32*8 + disp32 ]
add r32, [ r/m32*1 + r32 + disp32 ]
add r32, [ r/m32*2 + r32 + disp32 ]
add r32, [ r/m32*4 + r32 + disp32 ]
add r32, [ r/m32*8 + r32 + disp32 ]
add r/m32, r32
add [ disp32 ], imm32
add [ disp32 ], r32
add [ r/m32 ], imm32
add [ r/m32 + disp8 ], imm32
add [ r/m32 + disp32 ], imm32
add [ r/m32*1 + disp32 ], imm32
add [ r/m32*2 + disp32 ], imm32
add [ r/m32*4 + disp32 ], imm32
add [ r/m32*8 + disp32 ], imm32
add [ r/m32*1 + r32 + disp32 ], imm32
add [ r/m32*2 + r32 + disp32 ], imm32
add [ r/m32*4 + r32 + disp32 ], imm32
add [ r/m32*8 + r32 + disp32 ], imm32
add [ r/m32 + disp8 ], r32
add [ r/m32 + disp32 ], r32
add [ r/m32*1 + disp32 ], r32
add [ r/m32*2 + disp32 ], r32
add [ r/m32*4 + disp32 ], r32
add [ r/m32*8 + disp32 ], r32
add [ r/m32*1 + r32 + disp32 ], r32
add [ r/m32*2 + r32 + disp32 ], r32
add [ r/m32*4 + r32 + disp32 ], r32
add [ r/m32*8 + r32 + disp32 ], r32
```

4.2 jz/jnz/jmp

You must implement the following:

```
jz rel8
jz rel32
jnz rel8
jnz rel32
jmp rel8
jmp rel32
jmp r/m32
jmp [ disp32 ]
jmp [ r/m32 + disp8 ]
jmp [ r/m32 + disp32 ]
jmp [ r/m32*1 + disp32 ]
jmp [ r/m32*2 + disp32 ]
jmp [ r/m32*4 + disp32 ]
jmp [ r/m32*8 + disp32 ]
jmp [ r/m32*1 + r32 + disp32 ]
jmp [ r/m32*2 + r32 + disp32 ]
jmp [ r/m32*4 + r32 + disp32 ]
jmp [ r/m32*8 + r32 + disp32 ]
```

4.3 movsd

Recall that the "**d**" in "**movsd**" refers to the data size. In this case, it is a DWORD or 32-bit value. Thus, in the Intel Manual we are looking for "**movs m32, m32**".

Note: This is a string operation and NOT the "move scalar double-precision" operation.

4.4 repne cmpsd

Recall that the "**d**" in "**cmpsd**" refers to the data size. In this case, it is a DWORD or 32-bit value. Thus, in the Intel Manual we are looking for "**repne cmps m32, m32**".

4.5 retf/retn

For this instruction family (listed as just "**ret**" in the Intel Instruction Manual), you must implement the following:

```
retf
retf imm16
retn
retn imm16
```

*Note: "**retf**" refers to "return far" and "**retn**" refers to "return near."*

5 Sample Output

The following samples show how the input file is passed to your program and how output is to be formatted. The name of your program and the method by which it is invoked may be different depending on the language you use to implement it.

Example 1: No jumps

```
$ disasm -i nojump.o
00000000: 31C0              xor eax,eax
00000002: 01C8              add eax,ecx
00000004: 01D0              add eax,edx
00000006: 55                push ebp
00000007: 89E5              mov ebp,esp
00000009: 52                push edx
0000000A: 51                push ecx
0000000B: B844434241        mov eax,0x41424344
00000010: 8B9508000000      mov edx,[ebp+0x00000008]
00000016: 8B8D0C000000      mov ecx,[ebp+0x0000000c]
0000001C: 01D1              add ecx,edx
0000001E: 89C8              mov eax,ecx
00000020: 5A                pop edx
00000021: 59                pop ecx
00000022: 5D                pop ebp
00000023: C20800            retn 0x0008
```

Example 2: With conditional jump

```
$ disasm -i condjump.o
00000000: 55                push ebp
00000001: 89E5              mov ebp,esp
00000003: 52                push edx
00000004: 51                push ecx
00000005: 39D1              cmp ecx,edx
00000007: 740F              jz offset_00000018h
00000009: B844434241        mov eax,0x41424344
0000000E: 8B5508            mov edx,[ebp+0x00000008]
00000011: 8B4D0C            mov ecx,[ebp+0x0000000c]
00000014: 01D1              add ecx,edx
00000016: 89C8              mov eax,ecx
offset_00000018h:
00000018: 5A                pop edx
00000019: 59                pop ecx
0000001A: 5D                pop ebp
0000001B: C20800            retn 0x0008
```

6 Complete Opcode and Addressing Mode Requirements

Mnemonic/Syntax	Opcode	Addressing Modes
add eax, imm32	**0x05 id**	MODR/M Not Required
add r/m32, imm32	**0x81 /0 id**	00/01/10/11
add r/m32, r32	**0x01 /r**	00/01/10/11
add r32, r/m32	**0x03 /r**	00/01/10/11
and eax, imm32	**0x25 id**	MODR/M Not Required
and r/m32, imm32	**0x81 /4 id**	00/01/10/11
and r/m32, r32	**0x21 /r**	00/01/10/11
and r32, r/m32	**0x23 /r**	00/01/10/11
call rel32	**0xE8 cd**	Note: treat **cd** as **id**
call r/m32	**0xFF /2**	00/01/10/11
clflush m8	**0x0F 0xAE /7**	00/01/10 Note: **m8** can be a **[disp32]** only, a **[reg]**, a **[reg + disp8]**, or a **[reg + disp32]**. Addressing mode 11 is illegal.
cmp eax, imm32	**0x3D id**	MODR/M Not Required
cmp r/m32, imm32	**0x81 /7 id**	00/01/10/11
cmp r/m32, r32	**0x39 /r**	00/01/10/11
cmp r32, r/m32	**0x3B /r**	00/01/10/11
dec r/m32	**0xFF /1**	00/01/10/11
dec r32	**0x48 + rd**	MODR/M Not Required
idiv r/m32	**0xF7 /7**	00/01/10/11
inc r/m32	**0xFF /0**	00/01/10/11
inc r32	**0x40 + rd**	MODR/M Not Required

jmp rel8	**0xEB cb**	Note: treat **cb** as **ib**
jmp rel32	**0xE9 cd**	Note: treat **cd** as **id**
jmp r/m32	**0xFF /4**	00/01/10/11
jz rel8	**0x74 cb**	Note: treat **cb** as **ib**
jz rel32	**0x0f 0x84 cd**	Note: treat **cd** as **id**
jnz rel8	**0x75 cb**	Note: treat **cb** as **ib**
jnz rel32	**0x0f 0x85 cd**	Note: treat **cd** as **id**
lea r32, m	**0x8D /r**	00/01/10 Note: **m** can be a **[disp32]** only, a **[reg]**, a **[reg + disp8]**, or a **[reg + disp32].** Addressing mode 11 is illegal.
mov r32, imm32	**0xB8+rd id**	MODR/M Not Required
mov r/m32, imm32	**0xC7 /0 id**	00/01/10/11
mov r/m32, r32	**0x89 /r**	00/01/10/11
mov r32, r/m32	**0x8B /r**	00/01/10/11
movsd	**0xA5**	MODR/M Not Required
nop	**0x90**	MODR/M Not Required Note: this is really **xchg eax, eax**
not r/m32	**0xF7 /2**	00/01/10/11
or eax, imm32	**0x0D id**	MODR/M Not Required
or r/m32, imm32	**0x81 /1 id**	00/01/10/11
or r/m32, r32	**0x09 /r**	00/01/10/11
or r32, r/m32	**0x0B /r**	00/01/10/11
pop r/m32	**0x8F /0**	00/01/10/11
pop r32	**0x58 + rd**	MODR/M Not Required

push r/m32	**0xFF /6**	00/01/10/11
push r32	**0x50 + rd**	MODR/M Not Required
push imm32	**0x68 id**	MODR/M Not Required
repne cmpsd	**0xF2 0xA7**	MODR/M Not Required Note: **0xF2** is the **repne** prefix
retf	**0xCB**	MODR/M Not Required
retf imm16	**0xCA iw**	MODR/M Not Required Note: **iw** is a 16-bit immediate
retn	**0xC3**	MODR/M Not Required
retn imm16	**0xC2 iw**	MODR/M Not Required Note: **iw** is a 16-bit immediate
sub eax, imm32	**0x2D id**	MODR/M Not Required
sub r/m32, imm32	**0x81 /5 id**	00/01/10/11
sub r/m32, r32	**0x29 /r**	00/01/10/11
sub r32, r/m32	**0x2B /r**	00/01/10/11
test eax, imm32	**0xA9 id**	MODR/M Not Required
test r/m32, imm32	**0xF7 /0 id**	00/01/10/11
test r/m32, r32	**0x85 /r**	00/01/10/11
xor eax, imm32	**0x35 id**	MODR/M Not Required
xor r/m32, imm32	**0x81 /6 id**	00/01/10/11
xor r/m32, r32	**0x31 /r**	00/01/10/11
xor r32, r/m32	**0x33 /r**	00/01/10/11