MAKING OF A PROGRAMMING LANGUAGE

DAY 1

May 15,2021

Info at ccet.acm.org

About Speaker

Ishan Gambhir

 /gambhirishan2001  /CO18326

Computer Science Expert, Chegg India

Microsoft Technology Associate in Security Fundamentals (MTA)

Areas of Interest: Compiler Optimization, Computer Vision

Topics for Day 1

1. Setting goal: What is Compiler ?

2. Role of Assembler in Compilation Process (what is Assembler)

3. Intel Processors: 32 bit (X86) VS 64 bit

4. 8-bit registers VS 32-bit registers VS 64-bit registers

5. FASM (Flat Assembler)

What is Compiler?

What is Compiler ?

A PROGRAM USING SET of RULES to convert certain text from one form to another.

Compiler :an Analogy

Question in aptitude: If

XVJU = WMPK,

NTXC=ZQWL

Then what is VNT ?

Your brain as a Compiler

Question in aptitude: If

XVJU = WMPK,

NTXC=ZQWL

Then what is VNT ?

MZQ

Components of (Our Compiler)

Component A

+

Component B

=

Our Compiler

Component A

High level language

(ig)

Set of rules -> Assembly Language (intermediate language)

Component B

Assembly Language Set of rules-->

EXE File (Machine executable/ binary-language/ 1-0 file)

AT PRESENT:

Component A =

(we will code in python tomorrow)

Component B=

(FASM assembler)

Todays Focus : Component B

Component B =Assembler [

a program (a-mini compiler) that according to set of rules translate assembly language to 1-0 language (binary language)

]

OUR ASSEMBLER is FASM

A brand in the world of assemblers.

To understand ASSEMBLER

-> REGISTERS

(behaves like variables in C)

-> INSTRUCTION USED IN ASSEMBLER

(behaves like operators in C)

x86 Architecture

Basic Structure

Objectives

- You will understand how programs are stored in memory, and how they are executed by the processor.
- You will learn about the x86 32-bit registers.

Storing programs in memory

- The program to be run by the processor is written in memory (In RAM).
  - The memory is “made of” bits.
  - A **byte** is a set of 8 consecutive bits.
  - A byte is the basic information unit in x86 processors.
  - Usually a byte will be represented as two hexadecimal digits.
  - Hexadecimal digit (represents 4 bits) is sometimes called a **nibble**.
- Example for a simple program (represented in base 16):
  - 89 C1 01 C9 01 C1
  - Interpretation:
    - 89 C1 : mov ecx,eax
    - 01 C9 : add ecx,ecx
    - 01 C1 : add ecx,eax

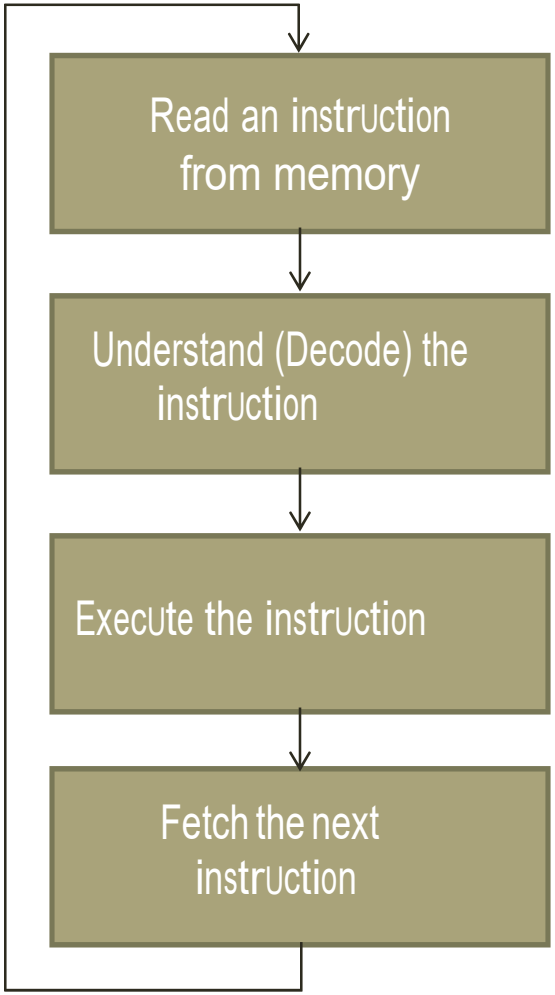
Storing programs in memory

- The program to be run by the processor is written in memory (In RAM).
  - The memory is "made of" bits.
  - A **byte** is a set of 8 consecutive bits.
  - A byte is the basic information unit in x86 processors.
  - Usually a byte will be represented as two hexadecimal digits.
  - Hexadecimal digit (represents 4 bits) is sometimes called a **nibble**.
- Example for a simple program (represented in base 16):
  - 89 C1 01 C9 01 C1
  - Interpretation:
    - 89 C1 : mov ecx,eax
    - 01 C9 : add ecx,ecx
    - 01 C1 : add ecx,eax

x86 Instructions

- An Instruction in the x86 architecture represents some simple task.
  - Examples: Addition, subtraction, moving dataetc.
- Every instruction is encoded as one or more bytes.
  - Instructions come in various sizes. There are very short instructions (one byte), and very long instructions (sometimes even 10 bytes or more).
  - The numeric representation of an instruction is also called **opcode** (**Op**eration **code**).
- The processor can only understand the numeric representation of the instructions.
- You don't have to remember the numeric representation of the instructions. There is a textual representation for us humans.
  - Eventually though, you might remember some of the numeric representations of the instructions.

Processor's operation cycle



The registers

- x86 processors have some very efficient internal places to store data. These are called **registers**.
  - Built inside the processor. (Not on the RAM).
  - Very efficient.
  - Very scarce – There are only a few.
  - Represent the internal state of the processor.
  - As usual, made of bits.
- We will learn about the names of existent registers. Only later we will learn about the things we can do with them.
- Don’t worry if in the beginning you don’t remember the names of all the registers.

Basic registers

- Basic registers, each is made of **32 bits**:
  - eax – Accumulator.
  - ebx – Base index.
  - ecx – Counter.
  - edx – Data register.
- In early x86 processors every register had a specific job for specific operation.
  - In recent processors (Mostly in protected and Long modes), registers became more general purpose.
  - The old roles of the registers can help you remember their names.
- The “e” stands for extended.

Register extensions

- The 32-bit registers are extensions of the old 16 bit registers.
  - Due to backwards compatibility.

<table>
<tr><td colspan="3">eax (32 bit)</td></tr>
<tr><td></td><td colspan="2">ax (16 bit)</td></tr>
<tr><td></td><td>ah (8 bit)</td><td>al (8 bit)</td></tr>
</table>

- ax is just another name for the lowest 16 bits of eax.
  - al, ah are just another names for bits 0:7 and bits 8:15 of eax respectively.
  - h stands for high, l stands for low.
- Example:
  - If eax contains 0xABCD1234, then:
    - ax contains 0x1234
    - ah contains 0x12. al contains 0x34.

Register extensions (Cont.)

eax (32 bit)		
	ax (16 bit)	
	ah (8 bit)	al (8 bit)

ebx (32 bit)		
	bx (16 bit)	
	bh (8 bit)	bl (8 bit)

ecx (32 bit)		
	cx (16 bit)	
	ch (8 bit)	cl (8 bit)

edx (32 bit)		
	dx (16 bit)	
	dh (8 bit)	dl (8 bit)

64 bit extensions (Longmode)

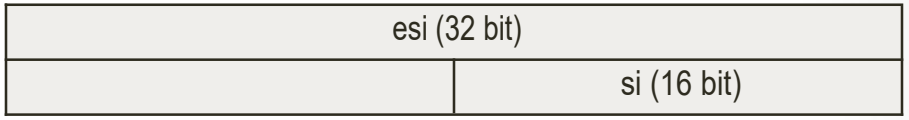
- The 32-bit registers were later extended again, to 64 bits:

<table>
<tr><td colspan="4">rax (64 bits)</td></tr>
<tr><td colspan="2"></td><td colspan="2">eax (32 bits)</td></tr>
<tr><td colspan="3"></td><td>ax (16 bits)</td></tr>
<tr><td colspan="3"></td><td>ah (8)</td><td>al (8)</td></tr>
</table>

- We are mostly going to deal with 32-bits registers in this course.
  - Almost every 32-bit register has a 64-bit equivalent.

More registers

- Index registers:
  - esi – Source Index register.
  - edi – Destination Index register.



- Instruction Pointer:
  - eip.

- Flags register.
- Stack pointers:
  - esp – Stack Pointer.
  - ebp – Base Pointer.

- There are even more registers.

Summary

- **Programs** are just a bunch of bytes.
- The processor can read bytes and interpret those as a program.
- There are places to store data inside the processor called **registers**. They represent the inner state of the processor.
  - Registers (In this course) are of size 32 bits.
  - Big registers are consisted of smaller registers, due to backwards compatibility.

- Don’t worry if you don’t remember the names of the registers. We are going to learn more about them in the following lectures.

First Instructions

X86 ARCHITECTURE

Objectives

You will learn about some basic x86 instructions.

Basic data manipulation.

- MOV

Simple Arithmetic.

- ADD
- SUB

Basic Instructionsstructure

- x86 Instructions have numeric representation (Opcode) and textual representation.
- x86 instructions have the following structure:
  - Mnemonic, or shortcut, for the instruction's name.
  - Arguments. (Needed for the operation).
- Written like this:
  - Mnemonic arg1,arg2,arg3,…
  - Usually no more than 2 arguments. (Sometimes even no arguments at all).
- The arguments are somehow encoded into the numeric representation.

Encoding instructions

There is a computer program that translates the textual representation of an instruction into the numeric representation of the instruction.

- This program is called **Assembler**.

While the numeric representation is unique and agreed upon, there are different textual flavors (Syntaxes) to represent the instructions.

We are going to use the syntax of the **fasm flat assembler**. We will learn more about it later in detail.

MOV

The MOV instruction allows to “move” data.

MOV destination, source

Data is copied from source to destination.

Examples:

mov eax,8CBh

- Will store the number 0x8CB inside the 32-bit register eax.

mov ecx,edx

- Will copy the number inside edx to ecx. (32 bit copy).

mov si,cx

- Will copy the number inside cx to si. (16 bit copy).

Invalid example: mov 13h,ecx

- It is not possible to assign ecx into 13h.

Invalid Example: mov ecx,dh

- ecx is of size 32 bits, but dh is of size 8 bits. Sizes don’t match.

MOV - Example

We make a table of the effects of various MOV instructions on eax, ecx and edx.

Instruction	eax	ecx	edx
	????????	????????	????????
mov eax, 3h			
mov edx, ABh			
mov edx, edx			
mov ecx, edx			
mov edx, eax			

MOV - Example

We make a table of the effects of various MOV instructions on eax, ecx and edx.

Instruction	eax	ecx	edx
	????????	????????	????????
mov eax, 3h	00000003	????????	????????
mov edx, ABh			
mov edx, edx			
mov ecx, edx			
mov edx, eax			

MOV - Example

We make a table of the effects of various MOV instructions on eax, ecx and edx.

Instruction	eax	ecx	edx
	????????	????????	????????
mov eax, 3h	00000003	????????	????????
mov edx, ABh	00000003	????????	000000AB
mov edx, edx			
mov ecx, edx			
mov edx, eax			

MOV - Example

We make a table of the effects of various MOV instructions on eax, ecx and edx.

Instruction	eax	ecx	edx
	????????	????????	????????
mov eax, 3h	00000003	????????	????????
mov edx, ABh	00000003	????????	000000AB
mov edx, edx	00000003	????????	000000AB
mov ecx, edx			
mov edx, eax			

MOV - Example

We make a table of the effects of various MOV instructions on eax, ecx and edx.

Instruction	eax	ecx	edx
	????????	????????	????????
mov eax, 3h	00000003	????????	????????
mov edx, ABh	00000003	????????	000000AB
mov edx, edx	00000003	????????	000000AB
mov ecx, edx	00000003	000000AB	000000AB
mov edx, eax			

MOV - Example

We make a table of the effects of various MOV instructions on eax, ecx and edx.

Instruction	eax	ecx	edx
	????????	????????	????????
mov eax, 3h	00000003	????????	????????
mov edx, ABh	00000003	????????	000000AB
mov edx, edx	00000003	????????	000000AB
mov ecx, edx	00000003	000000AB	000000AB
mov edx, eax	00000003	000000AB	00000003

MOV –Example (Cont.)

We make a table of the effects of various MOV instructions on eax, ecx and their partial counterparts.

Instruction	eax	ecx
	????????	????????
mov ax,9Ch		
mov eax,DDDD1234h		
mov cl,E5h		
mov ah,cl		

MOV –Example (Cont.)

We make a table of the effects of various MOV instructions on eax, ecx and their partial counterparts.

Instruction	eax	ecx
	????????	????????
mov ax,9Ch	????009C	????????
mov eax,DDDD1234h		
mov cl,E5h		
mov ah,cl		

MOV –Example (Cont.)

We make a table of the effects of various MOV instructions on eax, ecx and their partial counterparts.

Instruction	eax	ecx
	????????	????????
mov ax,9Ch	????009C	????????
mov eax,DDDD1234h	DDDD1234	????????
mov cl,E5h		
mov ah,cl		

MOV –Example (Cont.)

We make a table of the effects of various MOV instructions on eax, ecx and their partial counterparts.

Instruction	eax	ecx
	????????	????????
mov ax,9Ch	????009C	????????
mov eax,DDDD1234h	DDDD1234	????????
mov cl,E5h	DDDD1234	??????E5
mov ah,cl		

MOV –Example (Cont.)

We make a table of the effects of various MOV instructions on eax, ecx and their partial counterparts.

Instruction	eax	ecx
	????????	????????
mov ax,9Ch	????009C	????????
mov eax,DDDD1234h	DDDD1234	????????
mov cl,E5h	DDDD1234	??????E5
mov ah,cl	DDDDE534	??????E5

MOV –Example (Cont.)

We make a table of the effects of various MOV instructions on eax, ecx and their partial counterparts.

Instruction	eax	ecx
	????????	????????
mov ax,9Ch	????009C	????????
mov eax,DDDD1234h	DDDD1234	????????
mov cl,E5h	DDDD1234	??????E5
mov ah,cl	DDDDE534	??????E5

ADD

The ADD instruction allows to add numbers.

ADD destination, source

← +

The result wraps around if larger than the size of the arguments.

Examples:

add eax,edx

- Adds the contents of eax and edx. Stores the result in eax. (← +).

add esi,11b

- Adds the number 11 = 3_{10} toesi. (← +3).

add dx,si

- Adds the contents of si to dx, and stores the result in dx. (← +). Note that this is a 16 bit addition.

Invalid example: add 532h,ecx

- 532h can not be the destination of the addition operation. (Where willthe result be stored?)

Invalid example: add bx,eax

- bx is of size 16 bit, but eax is of size 32 bit. Sizes don’tmatch.

ADD - Example

Instruction	esi	eax	ebx
	00000001	00000002	00000003
add eax,ebx			
add eax,eax			
mov esi,0FFFFFFFFh			
add ebx,esi			
add esi,eax			

ADD - Example

Instruction	esi	eax	ebx
	00000001	00000002	00000003
add eax,ebx	00000001	00000005	00000003
add eax,eax			
mov esi,0FFFFFFFFh			
add ebx,esi			
add esi,eax			

ADD - Example

Instruction	esi	eax	ebx
	00000001	00000002	00000003
add eax,ebx	00000001	00000005	00000003
add eax,eax	00000001	0000000A	00000003
mov esi,0FFFFFFFFh			
add ebx,esi			
add esi,eax			

ADD - Example

Instruction	esi	eax	ebx
	00000001	00000002	00000003
add eax,ebx	00000001	00000005	00000003
add eax,eax	00000001	0000000A	00000003
mov esi,0FFFFFFFFh	FFFFFFFF	0000000A	00000003
add ebx,esi			
add esi,eax			

ADD - Example

Instruction	esi	eax	ebx
	00000001	00000002	00000003
add eax,ebx	00000001	00000005	00000003
add eax,eax	00000001	0000000A	00000003
mov esi,0FFFFFFFFh	FFFFFFFF	0000000A	00000003
add ebx,esi	FFFFFFFF	0000000A	00000002
add esi,eax			

ADD - Example

Instruction	esi	eax	ebx
	00000001	00000002	00000003
add eax,ebx	00000001	00000005	00000003
add eax,eax	00000001	0000000A	00000003
mov esi,0FFFFFFFFh	FFFFFFFF	0000000A	00000003
add ebx,esi	FFFFFFFF	0000000A	00000002
add esi,eax	00000009	0000000A	00000002

ADD –Example (Cont.)

Addition of partial registers:

Instruction	edi	ecx	eax
	AB29FFFF	00000703	000000FF
add al,ch			
add di,cx			
mov edi,0AB29FFFFh			
add edi,ecx			

ADD –Example (Cont.)

Addition of partial registers:

Instruction	edi	ecx	eax
	AB29FFFF	00000703	000000FF
add al,ch	AB29FFFF	00000703	00000006
add di,cx			
mov edi,0AB29FFFFh			
add edi,ecx			

ADD –Example (Cont.)

Addition of partial registers:

Instruction	edi	ecx	eax
	AB29FFFF	00000703	000000FF
add al,ch	AB29FFFF	00000703	00000006
add di,cx	AB290702	00000703	00000006
mov edi,0AB29FFFFh			
add edi,ecx			

ADD –Example (Cont.)

Addition of partial registers:

Instruction	edi	ecx	eax
	AB29FFFF	00000703	000000FF
add al,ch	AB29FFFF	00000703	00000006
add di,cx	AB290702	00000703	00000006
mov edi,0AB29FFFFh	AB29FFFF	00000703	00000006
add edi,ecx			

ADD –Example (Cont.)

Addition of partial registers:

Instruction	edi	ecx	eax
	AB29FFFF	00000703	000000FF
add al,ch	AB29FFFF	00000703	00000006
add di,cx	AB290702	00000703	00000006
mov edi,0AB29FFFFh	AB29FFFF	00000703	00000006
add edi,ecx	AB2A0702	00000703	00000006

ADD –Example (Cont.)

Addition of partial registers:

Instruction	edi	ecx	eax
	AB29FFFF	00000703	000000FF
add al,ch	AB29FFFF	00000703	00000006
add di,cx	AB290702	00000703	00000006
mov edi,0AB29FFFFh	AB29FFFF	00000703	00000006
add edi,ecx	AB2A0702	00000703	00000006

Wraparound is done according to the size of arguments.

SUB

The SUB instruction subtracts numbers.

SUB destination,source.

← −

The result wraps around if needed.

- Equivalent to ← +(−),where− is found using the two's complement method.

Examples:

sub eax,edx

- Subtracts edx from eax, and stores the result in eax. (← −)

sub cl,dl

- Subtracts dl from cl. Stores the result inside cl. (← −).

sub esi,4h

- Subtracts 4 from esi, and stores the result back in esi. (← −4).

Invalid example: sub eax,dl

- eax is of size 32 bits. dl is of size 8 bits. Sizes mismatch.

Invalid example: sub 1Ah,dl

- It is impossible to store the result inside 1Ah. No such opcode exists.

SUB - Example

Instruction	eax	ebx	ecx
	0000001A	00000003	00000002
sub eax,ebx			
add eax,ebx			
sub ecx,ebx			
add ecx,eax			
sub cl,al			

SUB - Example

Instruction	eax	ebx	ecx
	0000001A	00000003	00000002
sub eax,ebx	00000017	00000003	00000002
add eax,ebx			
sub ecx,ebx			
add ecx,eax			
sub cl,al			

SUB - Example

Instruction	eax	ebx	ecx
	0000001A	00000003	00000002
sub eax,ebx	00000017	00000003	00000002
add eax,ebx	0000001A	00000003	00000002
sub ecx,ebx			
add ecx,eax			
sub cl,al			

SUB - Example

Instruction	eax	ebx	ecx
	0000001A	00000003	00000002
sub eax,ebx	00000017	00000003	00000002
add eax,ebx	0000001A	00000003	00000002
sub ecx,ebx	0000001A	00000003	FFFFFFFF
add ecx,eax			
sub cl,al			

SUB - Example

Instruction	eax	ebx	ecx
	0000001A	00000003	00000002
sub eax,ebx	00000017	00000003	00000002
add eax,ebx	0000001A	00000003	00000002
sub ecx,ebx	0000001A	00000003	FFFFFFFF
add ecx,eax	0000001A	00000003	00000019
sub cl,al			

SUB - Example

Instruction	eax	ebx	ecx
	0000001A	00000003	00000002
sub eax,ebx	00000017	00000003	00000002
add eax,ebx	0000001A	00000003	00000002
sub ecx,ebx	0000001A	00000003	FFFFFFFF
add ecx,eax	0000001A	00000003	00000019
sub cl,al	0000001A	00000003	000000FF

SUB - Example

Instruction	eax	ebx	ecx
	0000001A	00000003	00000002
sub eax,ebx	00000017	00000003	00000002
add eax,ebx	0000001A	00000003	00000002
sub ecx,ebx	0000001A	00000003	FFFFFFFF
add ecx,eax	0000001A	00000003	00000019
sub cl,al	0000001A	00000003	000000FF

Wraparound is done according to arguments size.

Summary

MOV copies data from place to place. ADD adds numbers.

SUB subtracts numbers.

Exercises

Some code reading and predicting the resulting values of registers.

Some code writing.

Make sure to solve everything before moving on.

- Very important for your understanding of the instructions and registers.