ASSEMBLY LANGUAGE

Program execution in any microprocessor system consists of fetching binary information from memory and decoding that information to determine the instruction represented.

Humans have great difficulty in writing programs directly in ***machine code***, the binary language understood by the microprocessor.

Its much easier for humans to remember a mnemonic such as **SUB AX,AX** rather than the corresponding machine code **2BC0**

Machine Level Programming

Programming a computer in its native language i.e. binary coded instructions.

However:

- Its very primitive - low level.
- Its very precise - no room for errors.
- Its very specific - machine (CPU) dependent.
- Its very detailed - registers, addresses, flags, addressing modes etc, etc.
- Idiosyncratic - dealing directly with the peculiarities of the computer system hardware.

Assembly Language

Most computer programs these days are written in high-level languages such as JAVA, PYTHON, VISUAL C++, etc.

Writing programs in any of these is quicker and easier (and therefore cheaper if you are paying someone to do it) than in the machines own language – **machine code.**

Assembly Language

There are occasions, however, when a high level language just cannot be used basically from the point of view of speed

In the remaining slides we will see:

- What's wrong with assembly language
- What's right with assembly language

What is Assembly Language?

A programming language in which each statement produces *exactly one* machine instruction

Human-readable machine code

- Machine specific

Symbols used for

- Addresses (registers,variables,…)
- Machine instructions

What's Wrong With Assembly Language? (1)

It's hard to learn

But so is learning any other programming language

Java, C, Python,…

Declarative programs such as: Prolog,…

- `mother(X,Y) :- female(X), child(X,Y).`

Functional programs such as : Miranda, …

- `length [] = 0`
- `length (x:xs) = 1 + length xs`

What's Wrong With Assembly Language? (2)

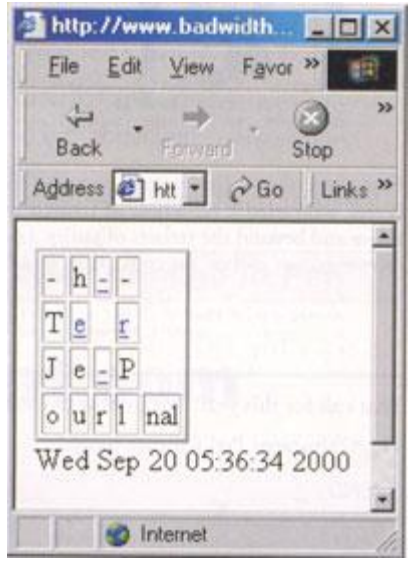
It's hard to read and understand

It can be:

```
SETHI %HI(I),%R1
LD [%R1+%LO(I)],%R1
SETHI %HI(J),%R2
LD [%R2+%LO(J)],%R2
```

But other languages can be too: An HTML sliding tile puzzle in 512 bytes.

Nemo, a redneck cracker from Georgia




```
$0=s#.*[/\\]##g;push@ARGV,"-h--Te_-JPerourl";$_=shift;@4=@1=@3=@2=@[=split(//);/ _/g;$;=pos;
$$=sub{eval"(\$$\\[$;\],\$$\\[$\"+$;\])=(\$$\\[$\"+$;\],\$$\\[$;\]);\ @\[\[$\"+$;\].=join\"\",\@$\ ;\$\\++"};$\++;$;--;
$"--;$;%4>0&&&$$;$"+=2;$;%4<3&& &$$;$"-=5;$;>3&&&$$;$"+=8;$;<12&&&$$;$\--;for(@[){s}(.)(.+)}
<TD><a href=\"$0?$2\ ">$1</a>}g;s@^(.)$@<TD>$1@g;s]>_]> ]g}for(0..3){@[[$_<<2]="<TR>".@[[$_<<2]};
$$=l ocaltime;print "Content-Type: text/html\n\n<html><body><table border=2>",@[,"<TD >nal</table>$$<p></body></html>
```

What's Wrong With Assembly Language? (3)

It's hard to debug and maintain

Any programming language can be hard to debug if you've not had much experience with it

Programs (in general) are hard to maintain

- "Code rot"

This is the point of learning about software engineering

What's Wrong With Assembly Language? (4)

It's hard to write

This is reasonably true

But the situation is improving:

- Dedicated IDE's for assembly language
- Standard libraries
  - Solve many common programming problems
  - E.g: *The UCR Standard Library for 80x86 Assembly Language Programmers*
  - Windows API

What's Wrong With Assembly Language? (5)

It takes a long time to write

This is also true

- (At least for inexperienced programmers)

Software engineers typically spend about 30% of their time writing code

- If writing in assembler takes twice as long, this will only increase the length of a project by about 30%

10% of the code accounts for 90% of program execution time

A *mixed* approach is very common

- Code the 10% in assembler, and the 90% in a high-level language

What's Wrong With Assembly Language? (6)

Compilers have eliminated the need for assembly language

Optimising compilers are *better* than humans at transforming a high-level language into machine code

But assembly programmers *don't work this way*

- They write their code *differently*, and so are able to get better performance

What's Wrong With Assembly Language? (7)

Machines are so fast we no longer need it

Computers are *never* fast enough

On any given machine, the fastest programs will always be written in assembler

- Graphics engines for games
- "Number-crunchers" (simulators, weather-forecasting,…)

What's Wrong With Assembly Language? (8)

If you need more speed, use a better algorithm (and not assembler)

Using a better algorithm is "common sense"

- It applies as much to writing in assembler as to writing in a high-level language

You can implement any algorithm in assembler

- But there are things you can do in assembler that are virtually impossible in a high-level language

What's Wrong With Assembly Language? (9)

Machines have so much memory there is no need to worry about saving space

Space is important

- Embedded systems (mobile phones, controllers,…)
- Desktop computers

The smaller the program, the cheaper the machine that can run it

Smaller programs also run faster

- They are less likely to be swapped out

What's Wrong With Assembly Language? (10)

It's not portable

Code written for one machine cannot (usually) be run on another

Code written for different operating systems may also be incompatible

- Linux
- Windows

Many high-level languages have this problem

- Java doesn't, because it's interpreted

What's Right With Assembly Language? (1)

Speed

Assembly language programs are generally the fastest programs around

With carefully-crafted assembler code, 5x speedups are not uncommon

- But to achieve this takes experience
- Need to know the organisation of data and instructions on the machine

What's Right With Assembly Language? (2)

Space

Assembly language programs are often about 50% smaller than their compiled counterparts

- This figure includes the space needed for the data items

Compilers tend to generate bulky code

Data structures are implemented using a small number of basic data types

- Human programmers are more flexible

What's Right With Assembly Language? (3)

Capability

High-level languages are an abstraction of the hardware

While simpler, they rarely take account of the special features of the machine

- E.g I/O operations

Anything you can do on the machine you can do in assembly language

What's Right With Assembly Language? (4)

Knowledge

- Knowing what the compiler turns your source code into can help you write better programs
- This applies to Java too

It's important for particular applications and occupations:

- Writing compilers
- Building embedded-systems
- Graphics Engines
- Writing device-drivers

What's Right With Assembly Language? (5)

Reverse engineering:

People working with malware and anti-virus software, rarely if ever have the access to original source code. Usually it's the compiled runnable code.

In situations where there is a need to reverse engineer these types of programs there is a need to disassemble the code and be able to interpret the assembler code to understand what the program is doing.

Summary

Studying assembly language exposes the machine

Writing assembly code is the only way to get a feel for the architecture of a machine

Writing assembler gives you an appreciation for the compiler