Tutorial: Instruction Set Architecture

(ISA)

Yul Williams, D.Sc.

Goals

- At the conclusion of this session, the student should be able to:
  - Identify levels of programming languages
  - Understand the processes of program assembly and compilation
  - Use various addressing modes to access memory and I/O

Outline

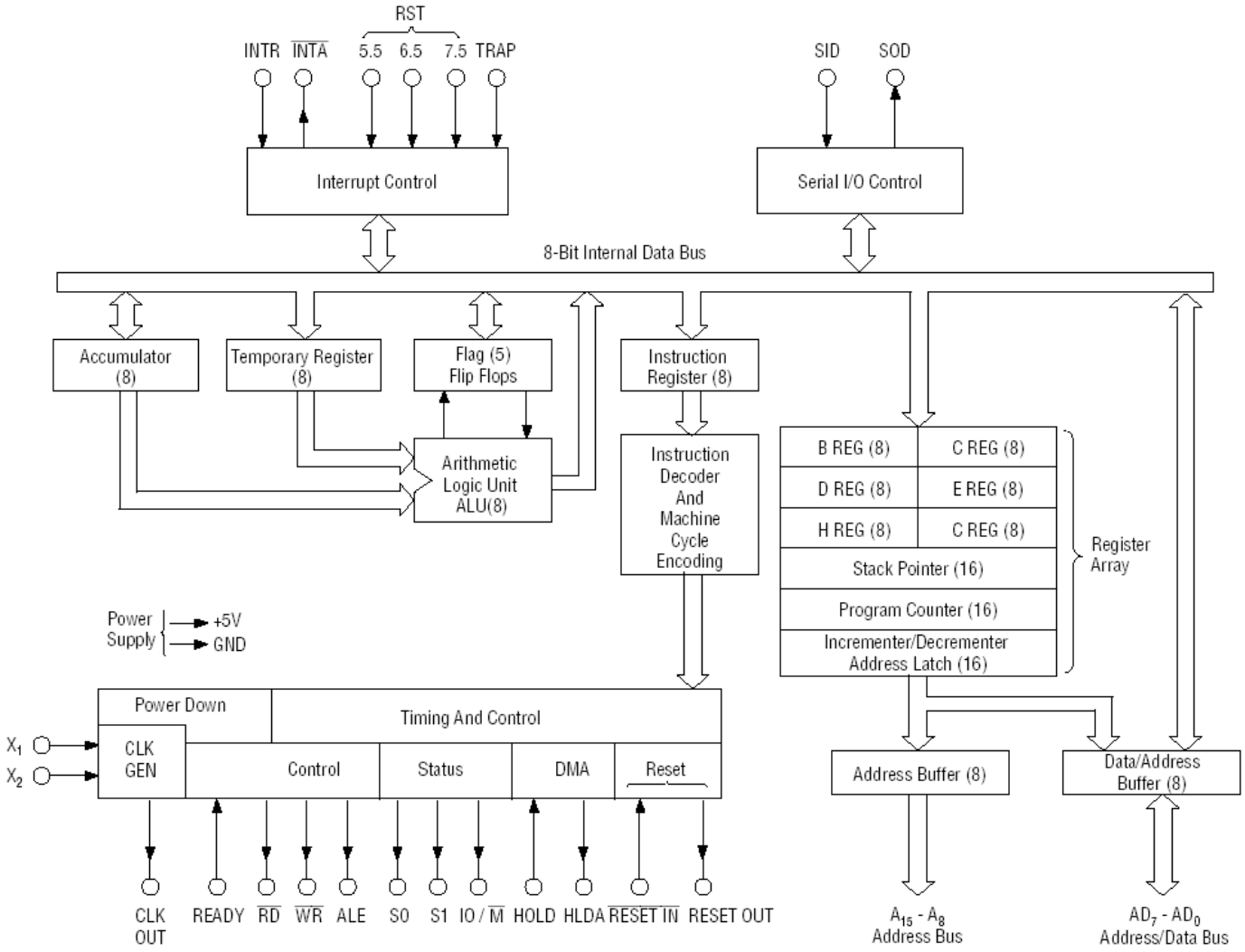
- What is an Instruction Set Architecture (ISA)?
- Levels of programming languages
- Compilation and assembly processes
- Assembly language instructions
- Addressing Modes
- Instruction Formats

What is an Instruction Set Architecture (ISA)?

Microprocessors:Background

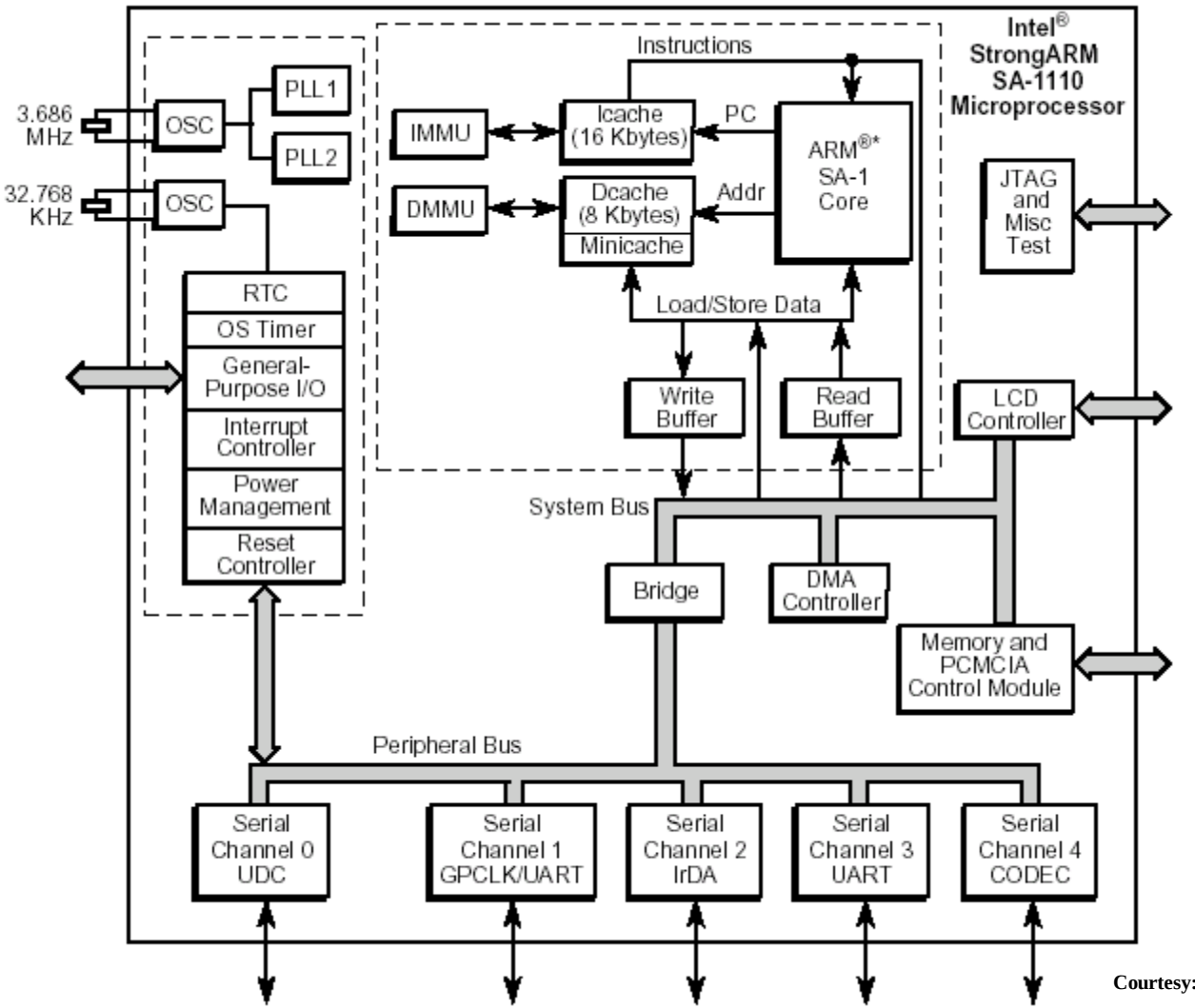
- Just as the automobile has an engine that is able to move it forward, the computer also has an engine
- The microprocessor is the centerpiece of the computer
  - It processes instructions and controls the various internal and peripheral computer components

Intel 8085 Microprocessor



Courtesy: OKI Semiconductors

Intel StrongARM Processor



Courtesy: Intel Corporation

Microprocessors Summary

- Contains essential features to perform:
  - Computations
  - I/O device interfacing
  - Simple and complex addressing
- Does not usually contain large on-chip memory (may contain cache)
- Requires off-chip peripherals to expand I/O capabilities
- We'll discuss more of this topic later…

Instruction Set Architecture

- The Instruction Set Architecture (ISA) is the visible set of instructions that the microprocessor can execute.
- It also defines the number and types of registers that a programmer of the microprocessor may directly access
  - Some registers in the microprocessor are not directly accessible by the programmer

Instructions

- The ISA instructions refer to the assembly language instructions
- Each microprocessor has its own set of assembly instructions. These instructions may not be executed by other processor types

Instruction set characteristics

- Fixed vs. variable length.
- Addressing modes.
- Number of operands.
- Types of operands.

Multiple implementations

- Successful architectures have several implementations:
  - varying clock speeds;
  - different bus widths;
  - different cache sizes;
  - etc.

Assembly language

- One-to-one with instructions (more or less).
- Basic features:
  - One instruction per line.
  - Labels provide names for addresses (usually in first column).
  - Instructions often start in later columns.
  - Columns run to end of line.

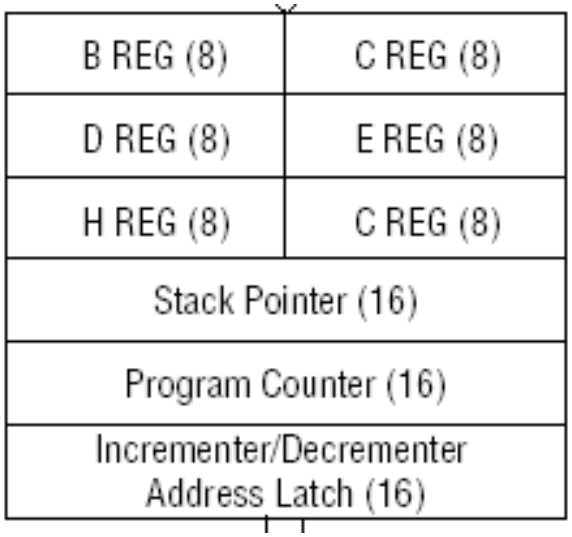
Programming model

- Programming model: registers visible to the programmer.
- Some registers are not visible (e.g IR and PC).

Registers

- The ISA describes the microprocessor registers that are accessible by the programmer
  - This description includes information about the register size (i.e. width) and the type of data element that it may contain
  - The ISA also indicates which specific instructions may access and manipulate the respective registers

Examples of Registers



These are some of the registers in the Intel 8085 microprocessor. We will discuss them in detail when we cover the topic of the 8085 ISA.

Levels of programming languages

Programming Language Levels

- Computer programming languages are usually divided into 3 categories:
  - High-level languages
  - Assembly languages
  - Machine languages
- Let's explore each level…

High Level Languages (HLLs)

- HLL's hide most of the details of the computer and operating system
- They provide a general means of programming the computer
- Many of these HLL's are platform independent because programs written in HLLs may be executed on a wide variety of computing platforms (i.e. computers)

Example HLLs

- C/C++
- FORTRAN
- Ada
- Pascal
- Modula, Modula-2 & Modula-3
- Java
- Lisp
- COBOL

Assembly Languages

- Assembly languages are unique to each microprocessor.
- They are categorized at a much lower level than the HLLs
- Assembly programs may not be executed on other computer systems with different microprocessors (unless the microprocessors are in the same architectural family and backward compatibility was built into the ISA)

Examples of Assembly Languages

- Intel 8085, 8086, 80286,…80486
- Intel Pentium
- Intel Itanium
- Motorola 68000, 68020 & 68040
- Motorola PowerPC
- SUN Sparc processor
- ARM
- Zilog eZ180

Backwards Compatibility

- Intel demonstrated the concept of backwards compatibility using the 80XXX and Pentium classes of microprocessors
- If you had a program that was written for the Intel 8086, it could be executed on all of the successor processors
- Each of the Intel processors since the 8086 have overlapping ISA features that support backward compatibility

Machine Languages

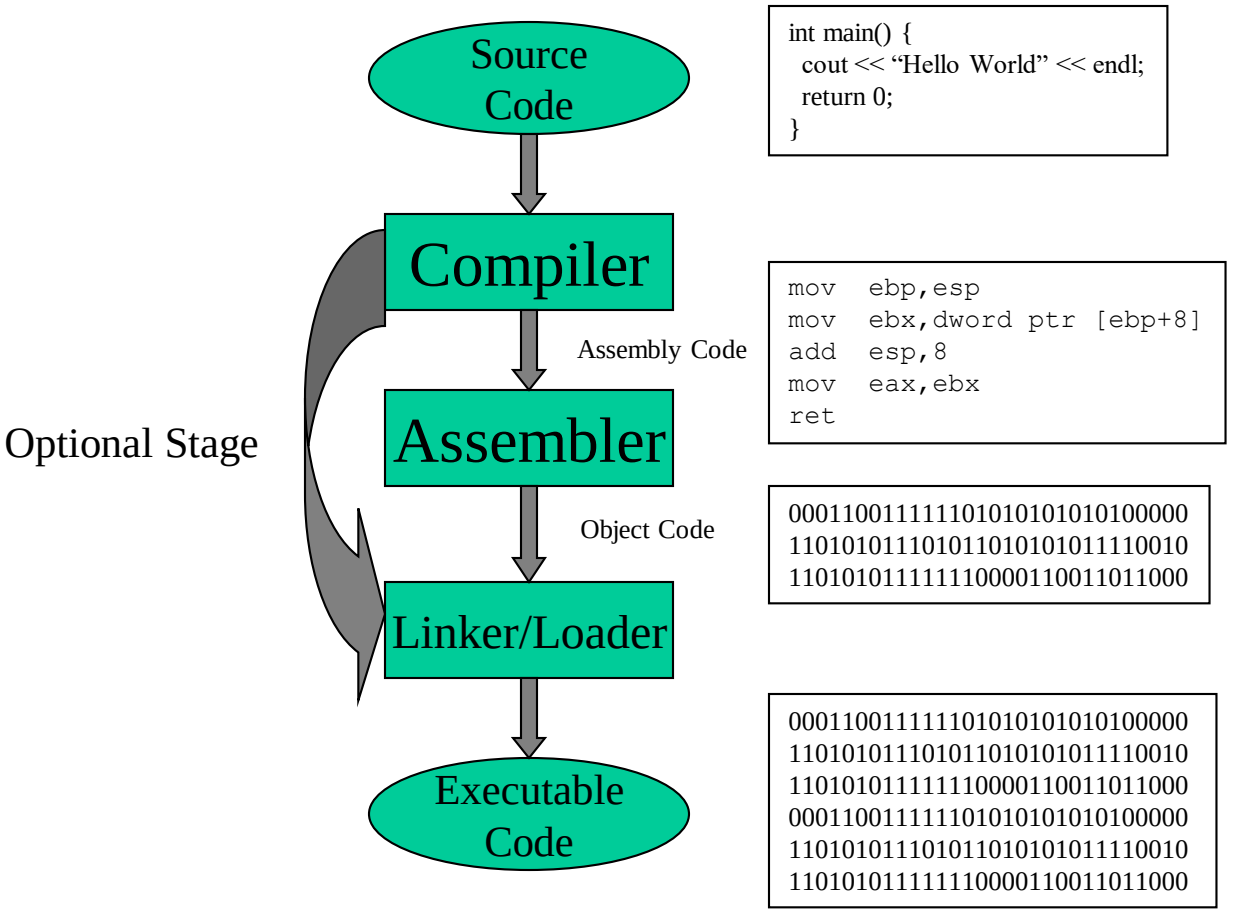
- The lowest level of programming languages is the class of machine languages
- These are basically binary encodings of the the machine instructions of specific microprocessors
- Programmers do not write machine language programs (okay…not as common these days!!)
- Machine language (or machine code) is automatically generated from the assembly or compilation processes

Compilation and Assembly Processes

Compiling Programs

- When a programmer writes a program in a high level language, that program must be converted from its source code format into its machine (object) code format for execution.
- The process by which HLL source code is converted into executable code is called compilation.

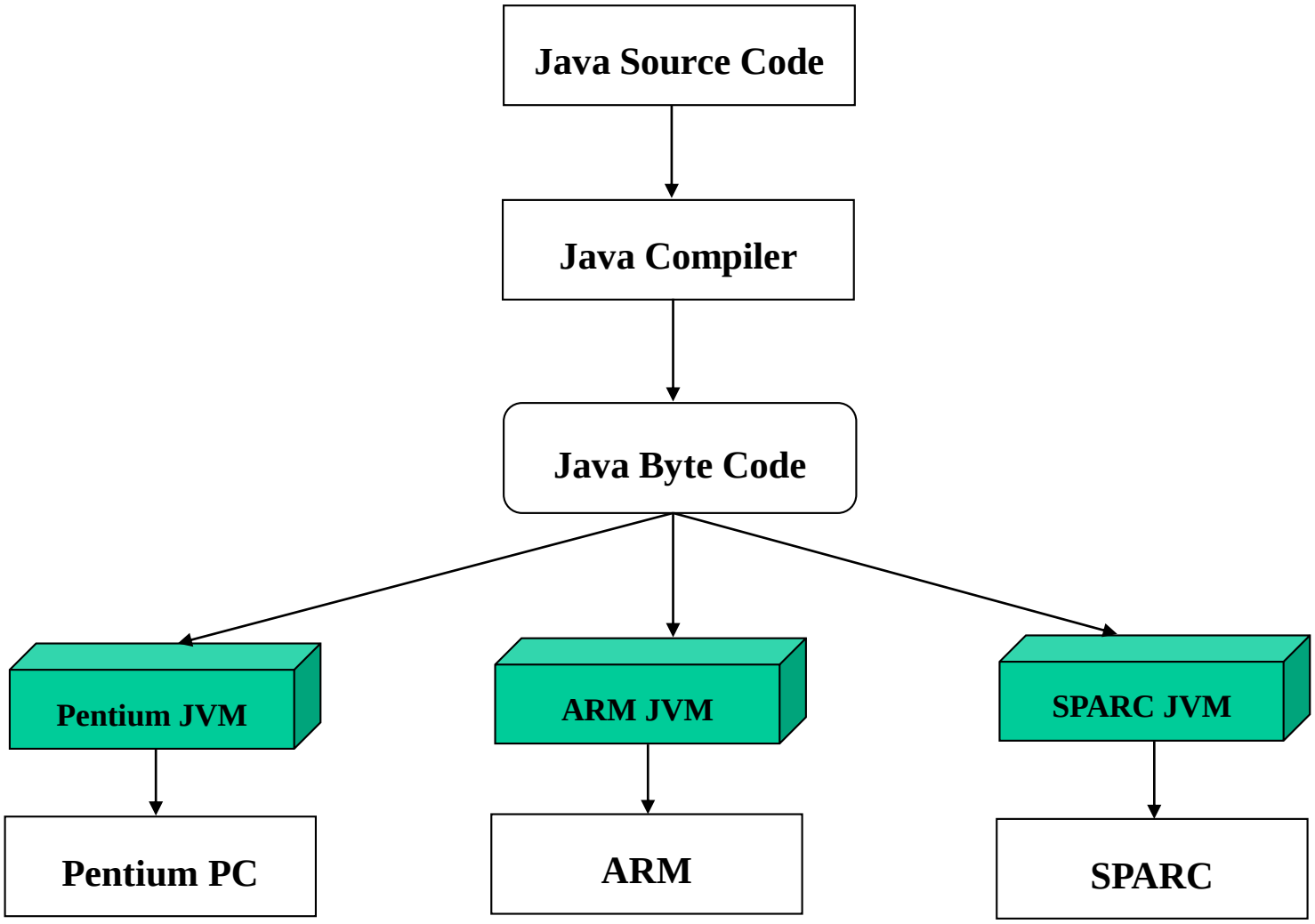
The Compilation Process



Compiling Java

- The Java language is executed on a virtual machine
- Each microprocessor that executes Java programs must be equipped with a Java Virtual Machine (JVM)
- Java programs (in general) are compiled into Java Byte Code
- Byte code compiled once can run on any machine with a standard JVM
- The JVM interprets the Java code and is machine independent

The Java Compilation Process



Assembling Programs

- When a programmer writes a program in assembly language, a specific assembler must be used to create the object code for that specific microprocessor
- The final executable file produced from this process can only be executed on a computer containing that specific type of microprocessor

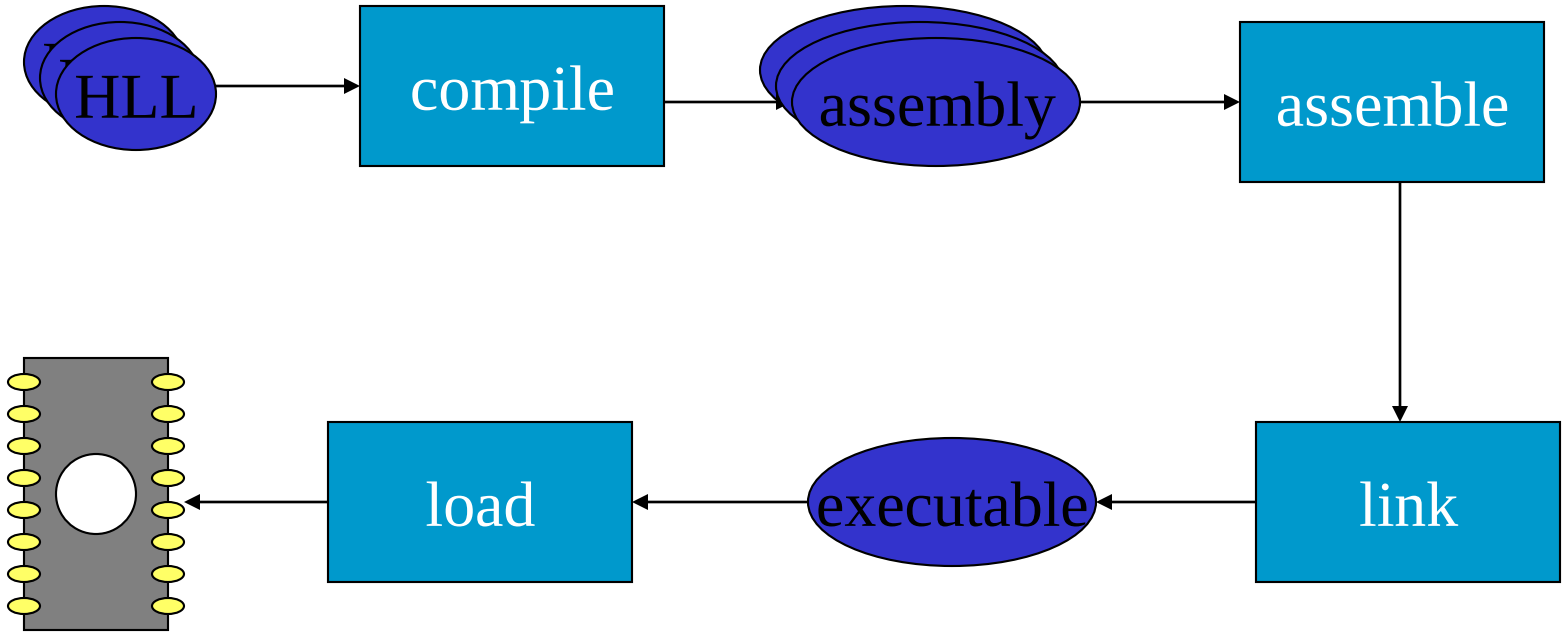
Assemblers

- Major tasks:
  - generate binary for assembly instructions;
  - translate labels into addresses;
  - handle pseudo-ops (data, etc.).
- Generally one-to-one translation.
- Assembly labels:

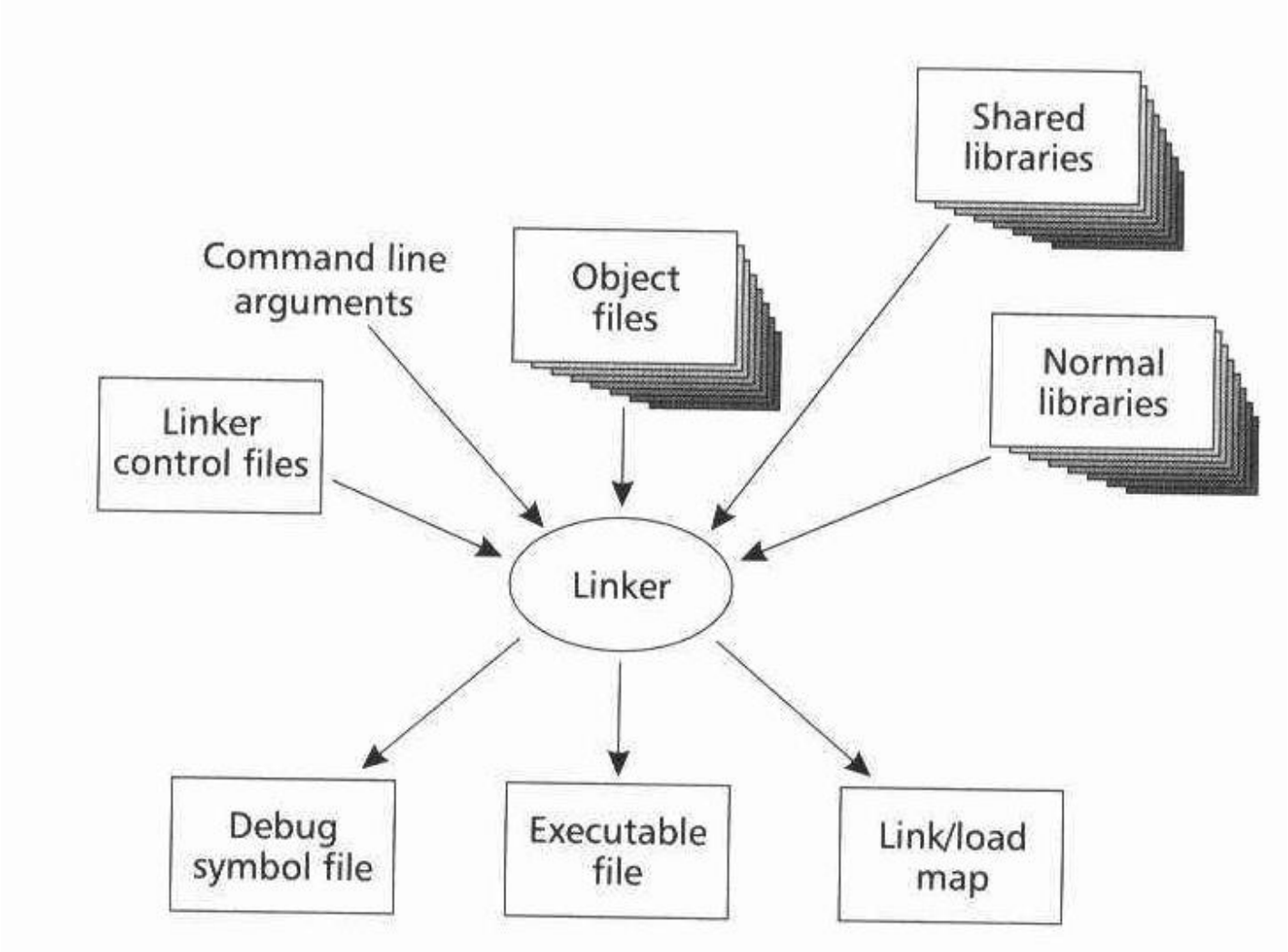
```
            ORG 100
label1      ADR r4,c
```

Assembly and linking

- Last steps in compilation:



The Linking Process



Assembly Language Instructions

Assembly Language Specifics

- Now that we have an understanding of how programs are converted from source code to executable code, we may explore the actual assembly instructions

Instruction Types

- For the purposes of this discussion, we will view assembly instructions in three categories:
  - Data Transfer Instructions
  - Data Operation Instructions
  - Program Control Instructions

Data Transfer Instructions

- Data transfer instructions are used to move data from one location to another.
- These instructions do not actually change/modify the data. Copies of the data remain in their original locations and are transferred to new destinations
- Instructions of this category include:
  - Load data from memory to the processor
  - Store data from the processor to memory
  - Move data within the processor
  - Input data to the processor
  - Output data from the processor

Data Operation Instructions

- Data operation instructions modify their data values
- They typically perform operations on one or more data values (called operands)
- Arithmetic instructions make up a large part of this category of instructions
  - This includes (add, subtract, multiply and divide)
- Logic instructions also fall into this category.
  - This includes (AND, OR, XOR and complement)
- Shift operations are also in this category

Program Control Instructions

- Program Control Instructions govern how the assembly language instructions are handled
  - Subroutines may be implemented in assembly language
  - Conditionals may be used for decision processing
  - Asynchronous events may also occur (such as interrupts)
  - Exception handling is another type of event that causes the flow of execution to follow a specific flow.
- Instructions such as Jump, Branch, Ret, Iret, and Halt are typical instruction in this category

Data Types

- Numeric Data
  - Integers
  - Floating Point
- Boolean Data
  - TRUE
  - FALSE
- Character Data
  - American Standard Code for Information Interchange (ASCII)
  - Extended Binary Coded Decimal Interchange Code (EBCDIC)
  - UNICODE (Used extensively by Java)

The ASCII Character Set

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	◘	BS	040	(	072	H	104	h
009	(tab)	HT	041	)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♫	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	►	DLE	048	0	080	P	112	p
017	◄	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	!!	DC3	051	3	083	S	115	s
020	¶	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↨	ETB	055	7	087	W	119	w
024	↑	CAN	056	8	088	X	120	x
025	↓	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[	123	{
028	(cursor right)	FS	060	<	092	\	124	|
029	(cursor left)	GS	061	=	093	]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	⌂

EBCDIC Characters

Characters with integer values from 64 – 249 in the ASCII standard are printable. There are some control characters intermingled wit the printable characters. The remaining 0-63 and 250-255 are non-printable control characters.

Unicode

What is Unicode?

Unicode provides a unique number for every character,
no matter what the platform,
no matter what the program,
no matter what the language.

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystem, Microsoft, Oracle, SAP, Sun, Sybase, Unisys and many others. Unicode is required by modern standards such as XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc.,

Source: www.unicode.org

Addressing Modes

Addressing Modes

- There are several methods that a microprocessor may generate addresses to access memory locations
- These methods are referred to as addressing modes

Addressing Modes

- Direct Mode
- Indirect Mode
- Register Direct and Register Indirect Modes
- Immediate Mode
- Implicit Mode
- Relative Mode
- Index and Base Address Modes

Direct Mode Example

;Semi-colons are used to indicate comments

;Load accumulator instruction

;This instruction grabs data from memory and stores it in an internal register (called the accumulator)

```
LDAC 10  ; Gets data from memory location 10 and stores the result in the
         ;CPU's accumulator
```

Indirect Mode

;Load accumulator instruction (indirect mode)

;This instruction grabs data from memory and stores it in an internal register (called the accumulator)

LDAC @ 10 **; Gets data from memory location 10 and uses the retrieved value as the ;address to access the desired data**

;30 is the value at memory location 10

;Load the value from memory location 30 into the CPU

Register Direct

;Load accumulator instruction (register direct mode)

;This instruction grabs data from a CPU register R and stores it in an internal register (called the accumulator)

LDAC R ; Gets data from the CPU register R and stores the result in the
;CPU's accumulator

Register Indirect Mode

;Load accumulator instruction (register indirect mode)

;This instruction grabs address information from a CPU register R and loads the data
;from the specified address

LDAC @ R **; Gets address from the CPU register R and loads the data**

;from the specified address

Immediate Mode

;Load accumulator instruction (direct mode)

;This instruction gets immediate data from the operand in the instruction and loads the it
;into the CPU's accumulator

LDAC #3 ; Gets data from the instruction operand and loads the data into the CPU

Implicit Mode

;Load accumulator instruction (implicit mode)

;This instruction gets data value from the stack and loads the data into the CPU's ;accumulator

LDAC ; Notice – no operands

Relative Mode

;Load accumulator instruction (relative mode)

;This instruction calculates the address of the data in memory by adding an offset value

;to the next address

LDAC $ 4 ; Gets address by adding an offset value to the next address and loads value into the
;CPU's accumulator register

Indexed Mode

;Load accumulator instruction (register indirect mode)

;This instruction gets a value from an index register X

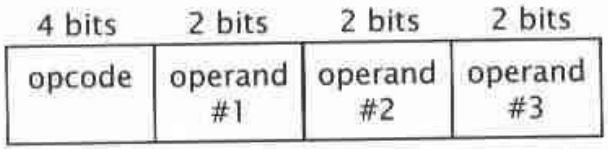
;It adds the data to an offset value to derive the desired address

LDAC 10(X) **; X = 10**

;Final address = 10 + 10 = 20

;Result is loading a data value from address location 20 into the CPU

Instruction Formats

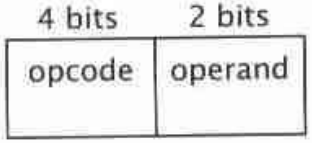


ADD *A,B,C* (*A=B+C*) 1010 00 01 10

(a)

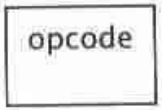
MOVE *A,B* (*A=B*) 1000 00 01
ADD *A,C* (*A=A+C*) 1010 00 10

(b)



LOAD *B* (*Acc=B*) 0000 01
ADD *C* (*Acc=Acc+C*) 1010 10
STORE *A* (*A=Acc*) 0001 00

(c)



PUSH *B* (Stack=*B*) 0101
PUSH *C* (Stack=*C,B*) 0110
ADD (Stack=*B+C*) 1010
POP *A* (*A*=stack) 1100

(d)

Practice Questions

1. What kind of instructions determine the flow of execution of a program.

2. The various methods of accessing memory are called _____.

3. What is the function of a linker?

4. What characteristic allows a program written for an Intel 80286 allows it to execute on an Intel Pentium 4 processor?

5. Can you explain the concept of a virtual machine?