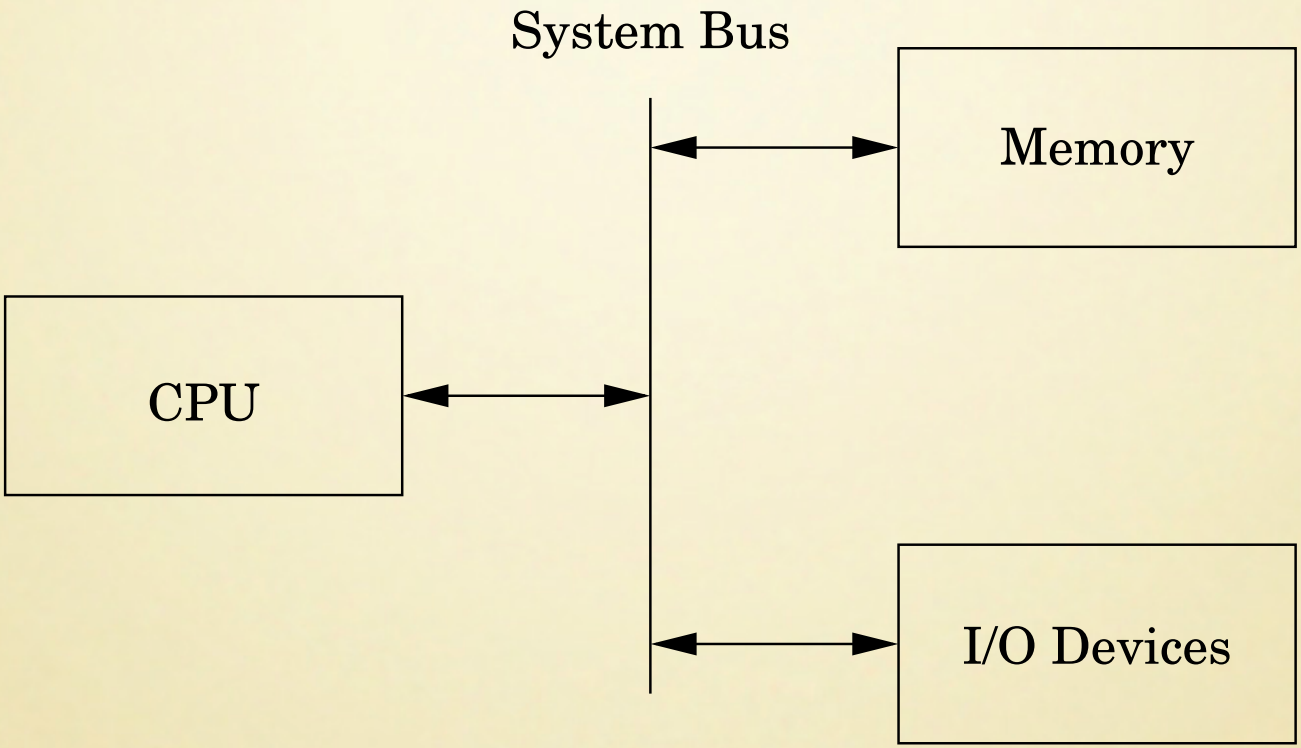
Modern Assembly Language Programming with the ARM processor

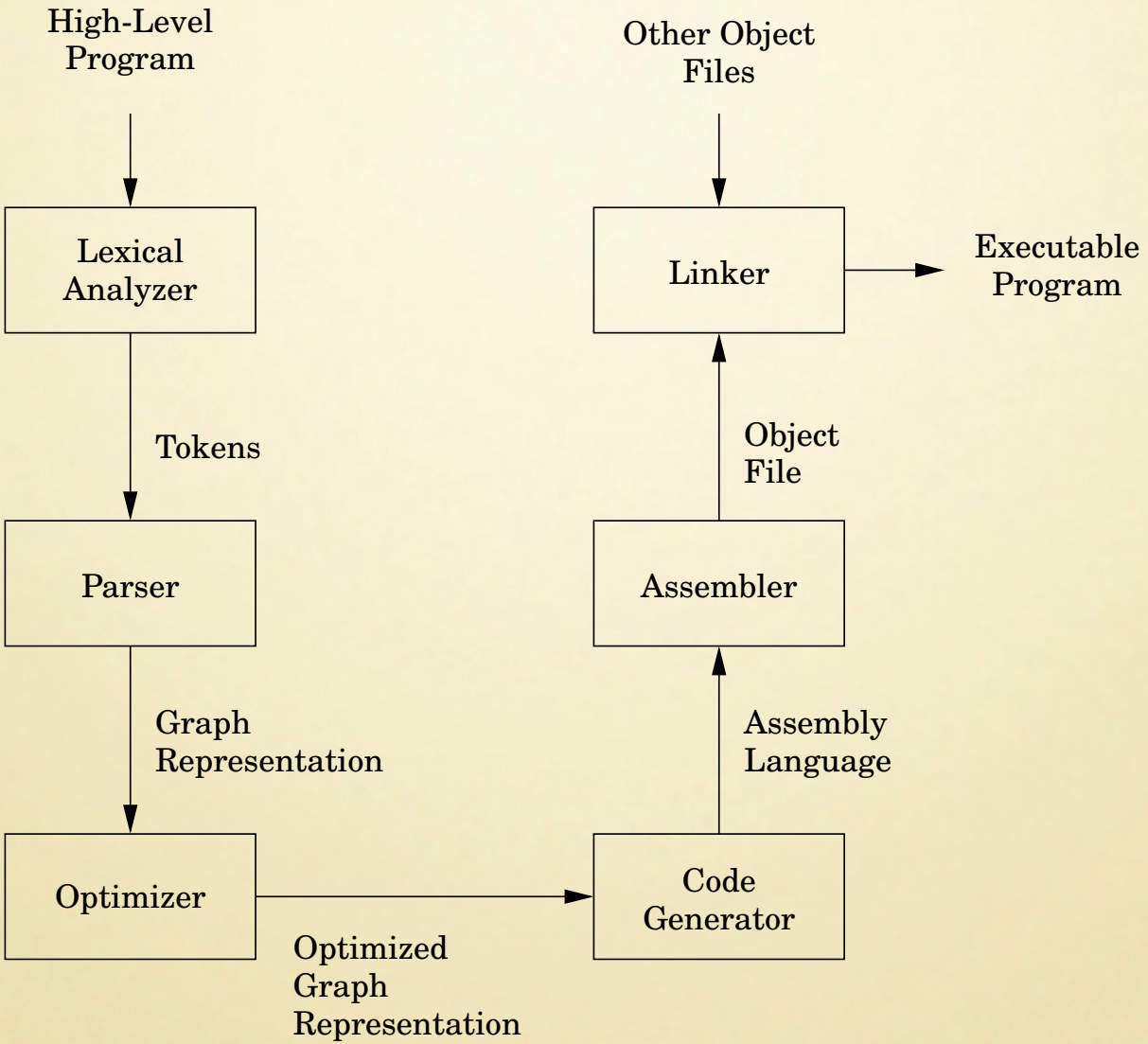
Chapter 1: Assembly as a Language

1. Introduction
2. Computers and Compilers
3. ARM Processor
4. Computer Data
   - Integers
   - Characters

Simplified Computer



Compilation Sequence



Reasons for Assembly Language

- writing the code generator for a compiler
- booting the computer
- interrupts
- low-level locking code for multi-threaded programs
- writing code for machines where no compiler exists
- the compiler cannot generate code that is efficient enough, or optimal
- the computer has very limited memory and the compiler cannot generate code that is small enough
- low-level access to architectural and processor features

More Reasons

- debugging
- writing the operating system
- SIMD instructions
- deeper understanding of the machine
- foundation for advanced topics such as
  - microcode
  - pipelining
  - cache issues
  - security

The ARM Processor

- load/store architecture
- reduced instruction set computer (RISC)
- orthogonal design
- relatively simple assembly language
- very large market share

Integers and Bases

Recall:

$$83483_{10} = 8 \times 10^4 + 3 \times 10^3 + 4 \times 10^2 + 8 \times 10^1 + 3 \times 10^0$$

Converting from an arbitrary base b to base 10 is easy:

$$\begin{aligned} 330425_7 &= 3 \times 7^5 + 3 \times 7^4 + 0 \times 7^3 + 4 \times 7^2 + 2 \times 7^1 + 5 \times 7^0 \\ &= 50421_{10} + 7203_{10} + 0_{10} + 196_{10} + 14_{10} + 5_{10} \\ &= 57839_{10} \end{aligned}$$

Base 10 to base b

Accomplished by repeated division by the base, b.
Example: Convert 8341_{10} to base 7

$$
\begin{array}{r}
1191 \\
7\overline{)8341} \\
7000 \\
\hline
1341 \\
700 \\
\hline
641 \\
630 \\
\hline
11 \\
7 \\
\hline
4
\end{array}
$$

The least significant digit is 4.

Base 10 to base b

Accomplished by repeated division by the base, b.

Example: Convert 8341_{10} to base 7

$$
\begin{array}{rcl}
1191 & \rightarrow & 170 \\
7\overline{)8341} & & 7\overline{)1191} \\
7000 & & 700 \\
\hline
1341 & & 491 \\
700 & & 490 \\
\hline
641 & & 1 \\
630 & & \\
\hline
11 & & \\
7 & & \\
\hline
4 & &
\end{array}
$$

The next digit is 1.

Base 10 to base b

Accomplished by repeated division by the base, b.

Example: Convert 8341_{10} to base 7

$$
\begin{array}{rcrcr}
1191 & \rightarrow & 170 & \rightarrow & 24 \\
7\overline{)8341} & & 7\overline{)1191} & & 7\overline{)170} \\
7000 & & 700 & & 140 \\
\hline
1341 & & 491 & & 30 \\
700 & & 490 & & 28 \\
\hline
641 & & 1 & & 2 \\
630 & & & & \\
\hline
11 & & & & \\
7 & & & & \\
\hline
4 & & & &
\end{array}
$$

The next digit is 2.

Base 10 to base b

Accomplished by repeated division by the base, b.

Example: Convert 8341_{10} to base 7

$$
\begin{array}{rcrcrcr}
1191 & \rightarrow & 170 & \rightarrow & 24 & \rightarrow & 3 \\
7\overline{)8341} & & 7\overline{)1191} & & 7\overline{)170} & & 7\overline{)24} \\
7000 & & 700 & & 140 & & 21 \\
\hline
1341 & & 491 & & 30 & & 3 \\
700 & & 490 & & 28 & & \\
\hline
641 & & 1 & & 2 & & \\
630 & & & & & & \\
\hline
11 & & & & & & \\
7 & & & & & & \\
\hline
4 & & & & & &
\end{array}
$$

The next digit is 3.

Base 10 to base b

Accomplished by repeated division by the base, b.

Example: Convert 8341_{10} to base 7

$$
\begin{array}{rcrcrcrcr}
 & & 1191 & \rightarrow & 170 & \rightarrow & 24 & \rightarrow & 3 & \rightarrow & 0 \\
7 &) & 8341 & & 7 &) & 170 & & 7 &) & 24 & & 7 &) & 3 \\
 & & 7000 & & & & 140 & & 21 & & \\
 & & \hline 1341 & & & & 30 & & 3 & & \\
 & & 700 & & & & 28 & & & & \\
 & & \hline 641 & & & & 2 & & & & \\
 & & 630 & & & & & & & & \\
 & & \hline 11 & & & & & & & & \\
 & & 7 & & & & & & & & \\
 & & \hline 4 & & & & & & & &
\end{array}
$$

The most significant digit is 3.

$$
\begin{aligned}
8341_{10} &= 33214_7 \\
&= 3 \times 7^4 + 3 \times 7^3 + 2 \times 7^2 + 1 \times 7^1 + 4 \times 7^0 \\
&= 7203 + 1029 + 98 + 7 + 4
\end{aligned}
$$

Complements

Nine's Complement Table

0	9
1	8
2	7
3	6
4	5
5	4
6	3
7	2
8	1
9	0

One's Complement Table

0	1
1	0

The nine's complement of 732 is 267.
The ten's complement of 732 is $267 + 1 = 268$.
The one's complement (a.k.a. logical complement) of 10100110 is 01011001.
The two's complement of 10100110 is $01011001 + 1 = 01011010$.

Interpreting Integers

Binary	Unsigned	Two's Complement	Sign Magnitude	Excess-127
00000000	0	0	0	-127
00000001	1	1	1	-126
⋮	⋮	⋮	⋮	⋮
01111110	126	126	126	-1
01111111	127	127	127	0
10000000	128	-128	-0	1
10000001	129	-127	-1	2
⋮	⋮	⋮	⋮	⋮
11111110	254	-2	-126	127
11111111	255	-1	-127	128

Representing Characters

Question:
The computer only stores numbers, so how do we represent textual data?

Answer:
We assign a number to each “glyph,” or character, in our language, and design hardware to display the glyphs.

If the printers, keyboards, displays, and other hardware agree on the assignment of glyphs to numbers, then everything works together.

In the early days of computing, each manufacturer had their own assignment of numbers to glyphs.

ASCII Non-printing characters

Binary	Oct	Dec	Hex	Abbr	Glyph	Name
000 0000	000	0	00	NUL	^@	Null character
000 0001	001	1	01	SOH	^A	Start of Header
000 0010	002	2	02	STX	^B	Start of Text
000 0011	003	3	03	ETX	^C	End of Text
000 0100	004	4	04	EOT	^D	End of Transmission
000 0101	005	5	05	ENQ	^E	Enquiry
000 0110	006	6	06	ACK	^F	Acknowledgment
000 0111	007	7	07	BEL	^G	Bell
000 1000	010	8	08	BS	^H	Backspace
000 1001	011	9	09	HT	^I	Horizontal Tab
000 1010	012	10	0A	LF	^J	Line feed
000 1011	013	11	0B	VT	^K	Vertical Tab
000 1100	014	12	0C	FF	^L	Form feed
000 1101	015	13	0D	CR	^M	Carriage return
000 1110	016	14	0E	SO	^N	Shift Out
000 1111	017	15	0F	SI	^O	Shift In
001 0000	020	16	10	DLE	^P	Data Link Escape
001 0001	021	17	11	DC1	^Q	Device Control 1 (oft. XON)
001 0010	022	18	12	DC2	^R	Device Control 2
001 0011	023	19	13	DC3	^S	Device Control 3 (oft. XOFF)
001 0100	024	20	14	DC4	^T	Device Control 4
001 0101	025	21	15	NAK	^U	Negative Acknowledgement
001 0110	026	22	16	SYN	^V	Synchronous idle
001 0111	027	23	17	ETB	^W	End of Transmission Block
001 1000	030	24	18	CAN	^X	Cancel
001 1001	031	25	19	EM	^Y	End of Medium
001 1010	032	26	1A	SUB	^Z	Substitute
001 1011	033	27	1B	ESC	^[	Escape
001 1100	034	28	1C	FS	^\	File Separator
001 1101	035	29	1D	GS	^]	Group Separator
001 1110	036	30	1E	RS	^^	Record Separator
001 1111	037	31	1F	US	^_	Unit Separator
111 1111	177	127	7F	DEL	^?	Delete

ASCII

Binary	Dec	Hex	Glyph	Binary	Dec	Hex	Glyph	Binary	Dec	Hex	Glyph
010 0000	32	20	␣	100 0000	64	40	@	110 0000	96	60	‘
010 0001	33	21	!	100 0001	65	41	A	110 0001	97	61	a
010 0010	34	22	"	100 0010	66	42	B	110 0010	98	62	b
010 0011	35	23	#	100 0011	67	43	C	110 0011	99	63	c
010 0100	36	24	$	100 0100	68	44	D	110 0100	100	64	d
010 0101	37	25	%	100 0101	69	45	E	110 0101	101	65	e
010 0110	38	26	&	100 0110	70	46	F	110 0110	102	66	f
010 0111	39	27	’	100 0111	71	47	G	110 0111	103	67	g
010 1000	40	28	(	100 1000	72	48	H	110 1000	104	68	h
010 1001	41	29	)	100 1001	73	49	I	110 1001	105	69	i
010 1010	42	2A	*	100 1010	74	4A	J	110 1010	106	6A	j
010 1011	43	2B	+	100 1011	75	4B	K	110 1011	107	6B	k
010 1100	44	2C	,	100 1100	76	4C	L	110 1100	108	6C	l
010 1101	45	2D	-	100 1101	77	4D	M	110 1101	109	6D	m
010 1110	46	2E	.	100 1110	78	4E	N	110 1110	110	6E	n
010 1111	47	2F	/	100 1111	79	4F	O	110 1111	111	6F	o
011 0000	48	30	0	101 0000	80	50	P	111 0000	112	70	p
011 0001	49	31	1	101 0001	81	51	Q	111 0001	113	71	q
011 0010	50	32	2	101 0010	82	52	R	111 0010	114	72	r
011 0011	51	33	3	101 0011	83	53	S	111 0011	115	73	s
011 0100	52	34	4	101 0100	84	54	T	111 0100	116	74	t
011 0101	53	35	5	101 0101	85	55	U	111 0101	117	75	u
011 0110	54	36	6	101 0110	86	56	V	111 0110	118	76	v
011 0111	55	37	7	101 0111	87	57	W	111 0111	119	77	w
011 1000	56	38	8	101 1000	88	58	X	111 1000	120	78	x
011 1001	57	39	9	101 1001	89	59	Y	111 1001	121	79	y
011 1010	58	3A	:	101 1010	90	5A	Z	111 1010	122	7A	z
011 1011	59	3B	;	101 1011	91	5B	[	111 1011	123	7B	{
011 1100	60	3C	<	101 1100	92	5C	\	111 1100	124	7C	|
011 1101	61	3D	=	101 1101	93	5D	]	111 1101	125	7D	}
011 1110	62	3E	>	101 1110	94	5E	^	111 1110	126	7E	~
011 1111	63	3F	?	101 1111	95	5F	_				

Encoding a String

Character	Binary	Hexadecimal	Decimal
H	01001000	48	72
e	01100101	65	101
l	01101100	6C	108
l	01101100	6C	108
o	01101111	6F	111
	00100000	20	32
W	01010111	57	87
o	01101111	6F	111
r	01110010	62	98
l	01101100	6C	108
d	01100100	64	100
NUL	00000000	00	0

Extensions to ASCII

- ISO 8859
  - Adds glyphs for many common languages.
- Unicode
  - Aims to provide glyphs for every character, in every written language, ever.
  - Still a work in progress, and there are some flaws.
  - Google's Noto font is probably the most comprehensive set at this time, but still lacks support for many languages.

ISO 8859

Name	Alias	Languages
ISO8859-1	Latin-1	Western European languages
ISO8859-2	Latin-2	Non-Cyrillic Central and Eastern European languages
ISO8859-3	Latin-3	Southern European languages and Esperanto
ISO8859-4	Latin-4	Northern European and Baltic languages
ISO8859-5	Latin/Cyrillic	Slavic languages that use a Cyrillic alphabet
ISO8859-6	Latin/Arabic	Common Arabic language characters
ISO8859-7	Latin/Greek	Modern Greek language
ISO8859-8	Latin/Hebrew	Modern Hebrew languages
ISO8859-9	Latin-5	Turkish
ISO8859-10	Latin-6	Nordic languages
ISO8859-11	Latin/Thai	Thai language
ISO8859-12	Latin/Devanagari	Never completed. Abandoned in 1997
ISO8859-13	Latin-7	Some Baltic languages not covered by Latin-4 or Latin-6
ISO8859-14	Latin-8	Celtic languages
ISO8859-15	Latin-9	Update to Latin-1 that replaces some characters. Most notably, it includes the euro symbol (€), which did not exist when Latin-1 was created
ISO8859-16	Latin-10	Covers several languages not covered by Latin-9 and includes the euro symbol (€)

Unicode and UTF-8

Unicode can be implemented by different character encodings. The most commonly used encodings are UTF-8 and UTF-16.

UTF-8 uses one byte for ASCII characters, all of which have the same code values in both UTF-8 and ASCII encoding.

Non-ASCII characters can be represented with up to four bytes.

Typical Virtual Memory Layout