CMPE 220

Class 2

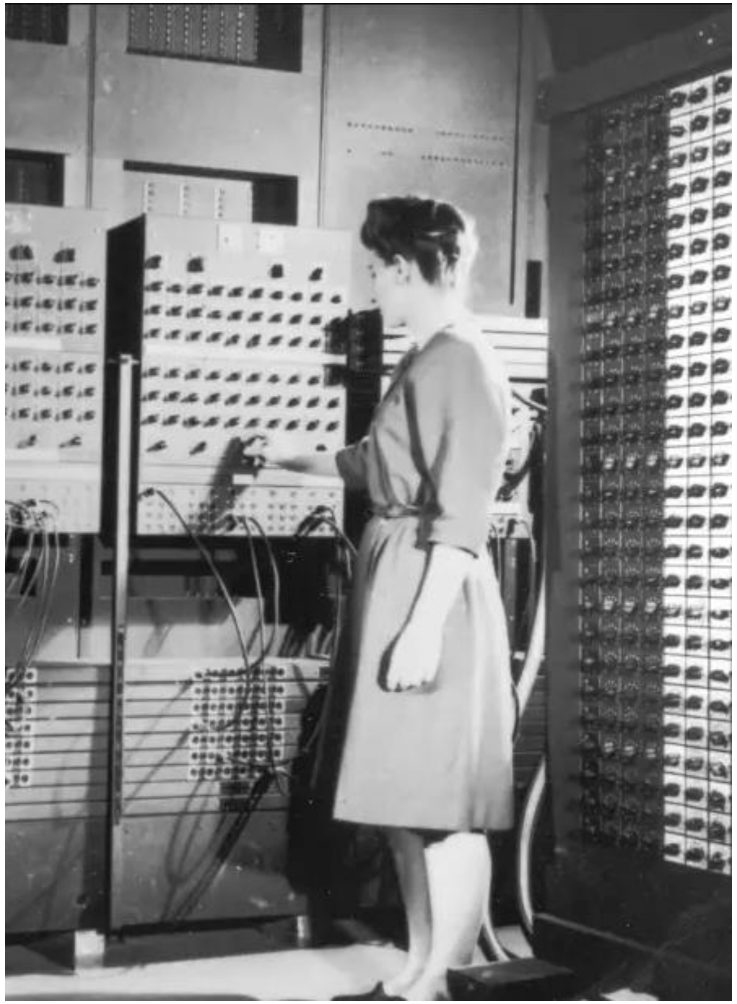
Computer Architecture

Binary Versus Decimal Arithmetic

Decimal	Binary	Binary Coded Decimal (BCD)
17405	0100001111111101	0000 0001 0111 0100 0000 0101
+ 9342	0010010001111110	0000 0000 1001 0011 0100 0010
= 26747	0110100001111011	0000 0010 0110 0111 0100 0111

Early History

- The first programmable general purpose computer was the *ENIAC* (Electronic Numerical Integrator and Calculator), developed by John Mauchley and J.Presper Eckert at the University of Pennsylvania.
- It performed *decimal arithmetic*.
- It took 2,800 microseconds to multiply two 10-digit decimal numbers… about 360 operations per second. (Desktop computers today are at least 1 billion times faster)
- It was delivered to the US army in 1945 and was used to computer artillery trajectories.



Early History - Continued

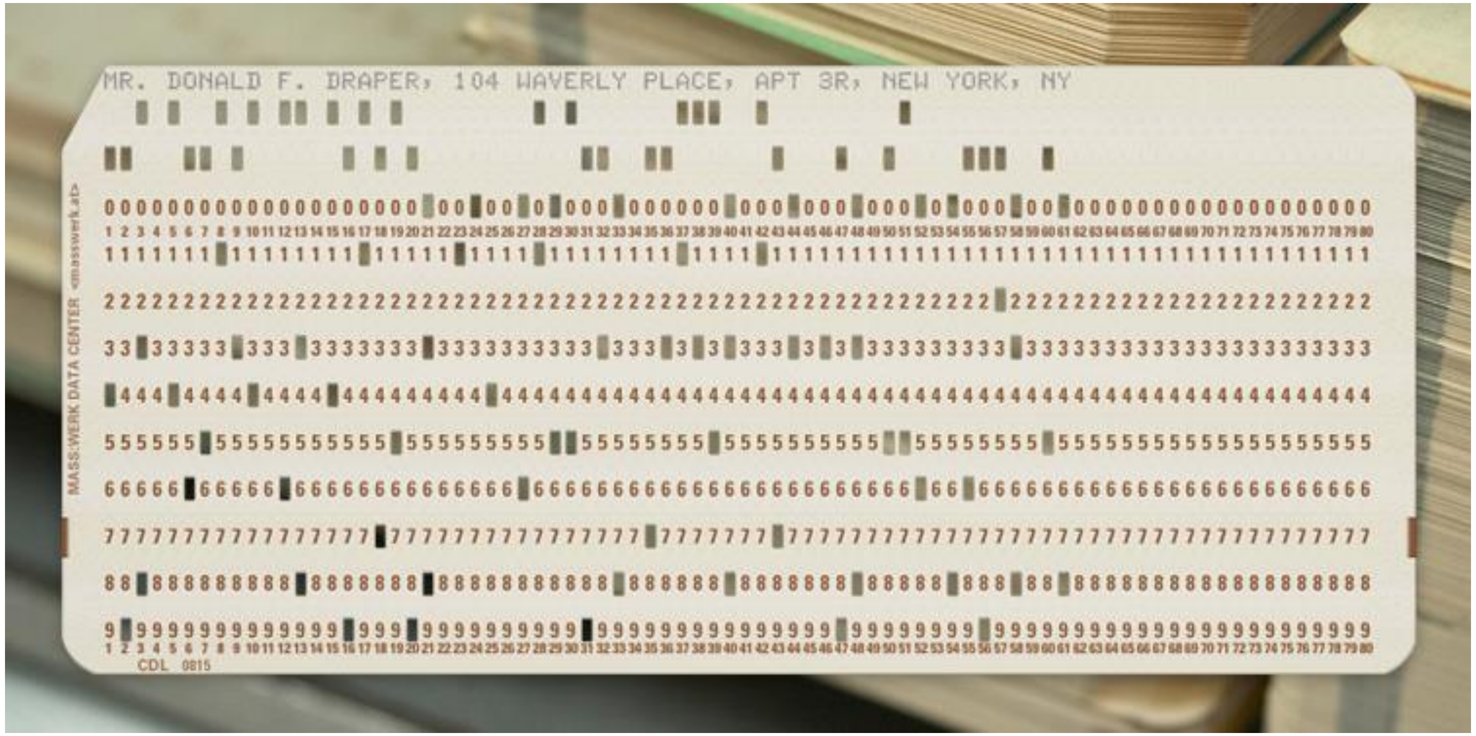
- ENIAC was followed by the *Univac* (UNIVersal Automatic Computer), the first successful *commercial* computer, in 1951.
- The Univac also used decimal arithmetic.
- The Univac had 1000 words of 12 characters each. An integer was represented as a + or – character, and 11 digit characters.
- These machines used vacuum tubes, which frequently burned out. They were "down" about 50% of the time.

Early History - Continued

- The ENIAC, UNIVAC, and other early computers used decimal arithmetic. Rather than representing numbers in binary, they represented numbers as a string of decimal digits, each digit encoded in binary (BCD).
- They had no provision for fixed point decimal or floating point numbers. If calculations involved non-integer numbers, the program needed to keep track of the decimal point independently.

Early History – Why Decimal Arithmetic?

- Comfort factor
- Numbers were entered as decimal; conversion was expensive
- Conversion caused rounding errors



BCD Arithmetic Today

- Decimal, or *Binary Coded Decimal* (BCD) arithmetic is still important:
  - It is *required* for some languages, such as COBOL
  - Ideal for financial computations, where conversion errors can be a problem.

    $0.01 = 0.00000010100011111011 (binary)

    0.00000010100011111011 = $0.0100002288818359375

- Integer, fixed-point, and floating point BCD arithmetic is supported in hardware on some processors.
- Other systems implement BCD arithmetic in software, via a *library*.

Binary Computers

- The first binary computer, the Z1, was developed in the 1930s, but was unreliable.
- The first widely successful binary computer was the IBM System/360, released in 1965.
  - It supported binary integer and binary floating point arithmetic, as well as decimal arithmetic.
  - It used a 32-bit word, which could hold four 8-bit characters.

Mini- and Micro-Computers

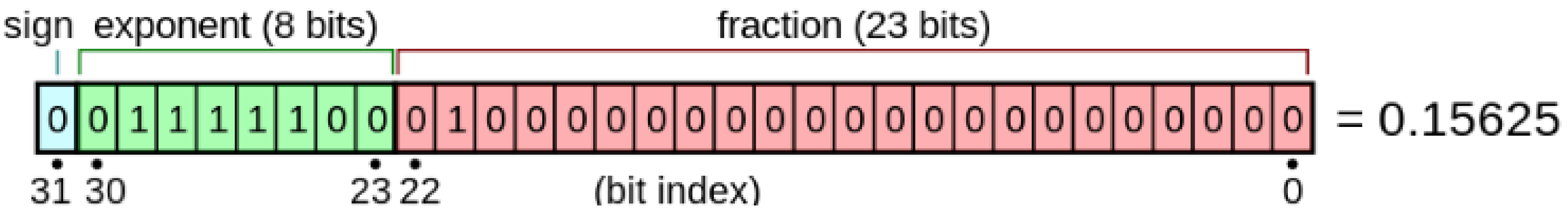
- The 1960s and 1970s saw the development of mini-computers (really just small, inexpensive computers) and micro-computers (based on single-chip CPUs).
- Mini-computers typically used 16-bit words, consisting of two 8-bit bytes. They supported multi-word formats for long integer and floating point arithmetic.
- Micro-computers, which appeared in the mid-70s, often used 8-bit words, but had the ability to do 16-bit arithmetic and 32-bit arithmetic.
- Integers were represented as 15 bits, plus a sign bit (+/- 32,767). Double-word integer arithmetic.

Precision and Portability

- In order for programs to be portable across platforms (hardware independent), arithmetic must work the same on any hardware.
- Language standards and test suites usually specify a *minimum* range and precision for each data type… but some systems support greater range and precision.
- A program may run correctly on a high-precision system, and fail when run on a lower precision system – even though both systems are standards-compliant. The programmers are inadvertently depending on higher precision than the standards.
- Some programs may even fail when running on *higher-precision* hardware.

Floating Point Arithmetic

- **IEEE 754:** a set of standards for binary and decimal floating point representation and arithmetic. The standard was established in 1985 and updated in 2008 and 2019.
- Each sub-standard specifies the number of bits (digits) and the exponent size.
  - **binary32:** 24 significant bits (including sign); 8 exponent bits



Floating Point Arithmetic - continued

- A floating point format consists of:
  - a base (also called *radix*) *b*, which is either 2 (binary) or 10 (decimal) in IEEE 754;
  - a precision *p*;
  - an exponent range from *emin* to *emax*, with *emin* = 1 − *emax* for all IEEE 754 formats
  - Specified representations for +infinity, -infinity, and NaN (not a number)
- The IEEE 754 floating-point standard also includes rules for rounding

Character Sets and Portability

- A character set is an encoding scheme for strings of text glyphs.
- Character sets are generally not tied to processor hardware.
- Character sets *are* tied to peripherals.
- Processors may have instructions for character set conversions.

Character Sets – ASCII (/ˈæski)

- ASCII (American Standard Code for Information Interchange)
- 7-bit ASCII (aka US ASCII): 0-127 - 1963
  - First 32 characters were reserved for *control functions*
  - Uppercase and lowercase English alphabet; 10 digits; punctuation and special characters (! @ # $ % etc)
  - Some programs used the 8th bit for other purposes!
- 8-bit (Extended) ASCII: 0-255
  - Adds many special and international characters: € † ™ £ ¶ Á á ä
  - Initially proprietary (system dependent)
  - ISO 8859 (ISO-8) defined several standard variants: Latin-1 (Western European), Latin-2 (Central European), Latin-3 (South European), Latin-4 (North European), Latin/Cyrillic, Latin/Arabic – 1987, 1999

7-Bit ASCII

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Character Sets – EBCDIC (/ˈɛbsɪdɪk/)

- **EBCDIC:** Extended Binary Coded Decimal Interchange Code - the "other" 8-bit character set
- Devised in 1963 for IBM mainframe computers and peripherals
- Encompassed BCD decimal characters
- ASCII and EBCDIC each contain characters not found in the other – such as "{" and "}" - making translations ambiguous and language support difficult
- Characters in EBCDIC are not in continuous alphabetical order, leading to potential portability issues:
  for (c='A';c<='Z';++c)
  - Steps through 26 characters if the character set is ASCII
  - Steps through 41 characters if the character set is EBCDIC

EBCDIC

NUL 00	SOH 01	STX 02	ETX 03	PF 04	HT 05	LC 06	DEL 07	GE 08	RLF 09	SMM 0A	VT 0B	FF 0C	CR 0D	SO 0E	SI 0F
DLE 10	DCI 11	DC2 12	TM 13	RES 14	NL 15	BS 16	IL 17	CAN 18	EM 19	CC 1A	CUI 1B	IFS 1C	IGS 1D	IRS 1E	IUS 1F
DS 20	SOS 21	FS 22	23	BYP 24	LF 25	ETB 26	ESC 27	28	29	SM 2A	CU2 2B	2C	ENQ 2D	ACK 2E	BEL 2F
30	31	SYN 32	33	PN 34	RS 35	UC 36	EOT 37	38	39	3A	CU3 3B	DC4 3C	NAK 3D	3E	SUB 3F
SP 40	41	42	43	44	45	46	47	48	49	¢ 4A	. 4B	< 4C	(4D	+ 4E	| 4F
& 50	51	52	53	54	55	56	57	58	59	! 5A	$ 5B	* 5C	) 5D	; 5E	¬ 5F
- 60	/ 61	62	63	64	65	66	67	68	69	¦ 6A	, 6B	% 6C	_ 6D	> 6E	? 6F
70	71	72	73	74	75	76	77	78	` 79	: 7A	# 7B	@ 7C	' 7D	= 7E	" 7F
80	a 81	b 82	c 83	d 84	e 85	f 86	g 87	h 88	i 89	8A	8B	8C	8D	8E	8F
90	j 91	k 92	l 93	m 94	n 95	o 96	p 97	q 98	r 99	9A	9B	9C	9D	9E	9F
A0	~ A1	s A2	t A3	u A4	v A5	w A6	x A7	y A8	z A9	AA	AB	AC	AD	AE	AF
{ C0	A C1	B C2	C C3	D C4	E C5	F C6	G C7	H C8	I C9	CA	CB	⑁ CC	CD	⑂ CE	CF
} D0	J D1	K D2	L D3	M D4	N D5	O D6	P D7	Q D8	R D9	DA	DB	DC	DD	DE	DF
\ E0	E1	S E2	T E3	U E4	V E5	W E6	X E7	Y E8	Z E9	EA	EB	⑀ EC	ED	EE	EF
0 F0	1 F1	2 F2	3 F3	4 F4	5 F5	6 F6	7 F7	8 F8	9 F9	| FA	FB	FC	FD	FE	EO FF

Character Sets – Unicode

- **Unicode:** unique, unified, universal encoding. An attempt to encode all of the world's character sets.
  - Unicode currently supports 137,994 glyphs and control characters.
- First draft standard was developed by Xerox and Apple in 1987.
- First complete standard was adopted in 1991.
- Actually a family of encoding standards: UTF-8, UTF-16, UTF-16BE, UTF-16LE, UTF-32, etc.
- **UTF-8:** an 8-bit variable-width encoding which *maximizes compatibility with ASCII*; used by 94% of all sites on the web
  - One byte for characters 0-127; up to 4 bytes for extended characters

Instruction Set Architecture

Elements of Machine Instructions

Memory Addressing

Memory

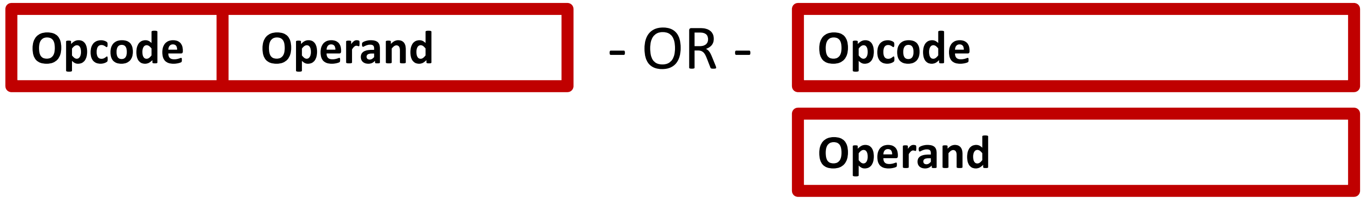
- Instructions and data are stored in memory
- Memory is external to the processor
- Accessing memory takes a significant amount of time

Registers

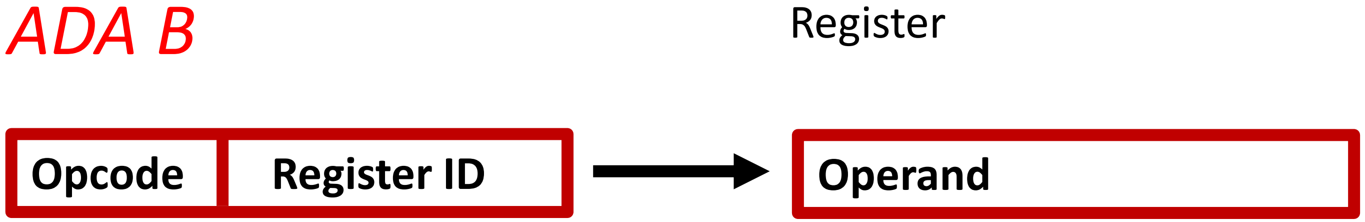
- Built into the processor
- Fast to access
- Limited in number
- Used directly by machine instructions:
  SBA 12 – (subtract 12 from the A register)

Addressing Modes

- Immediate
  SBA 12



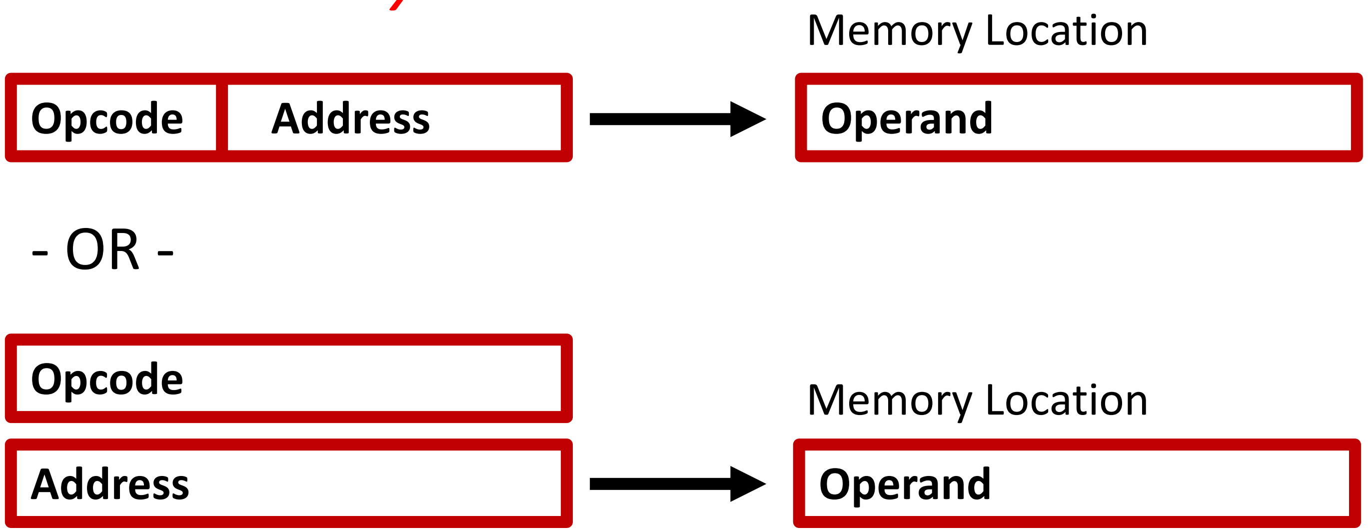
- Register
  ADA B



Addressing Modes - continued

- Direct

  LDA inventory

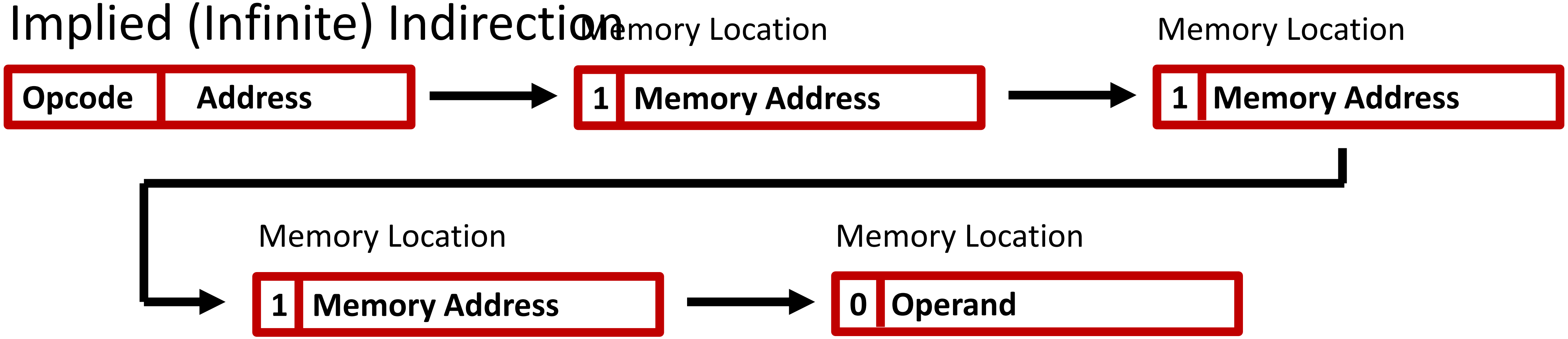


Addressing Modes - continued

- Indirect – Useful for arrays

  *LDA index**

- Implied (Infinite) Indirection



Addressing Modes - continued

- Register Indirect

- Register Indirect with Auto-increment/Auto-decrement
  - The register value is incremented or decremented after the memory fetch
  - Useful for fast processing of lists or arrays

Addressing Modes - continued

- Register Offset
  LDA array,X

Addressing Modes - continued

- Using Register Indirect or Register Offset with Auto-increment/Auto-decrement
- Easy in assembly language
  - Point register to list or array
  - Perform operations as register is incremented
- Difficult for compilers!

```
while (*from != '\0')
        *to++ = *from++;
```

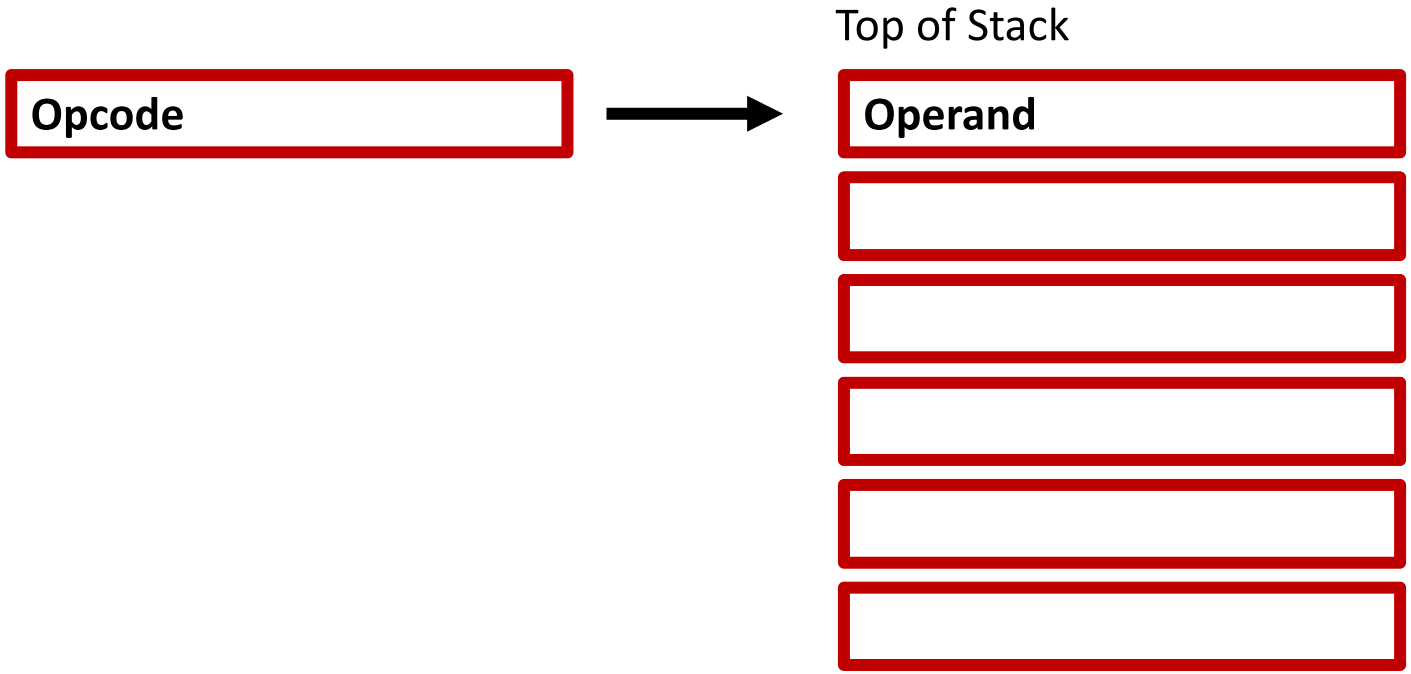
Addressing Modes - continued

- Relative or PC (Program Counter) Relative
  JMP loop

- Typically used for branch or conditional branch instructions

Addressing Modes - continued

- ***** Stack Addressing *****



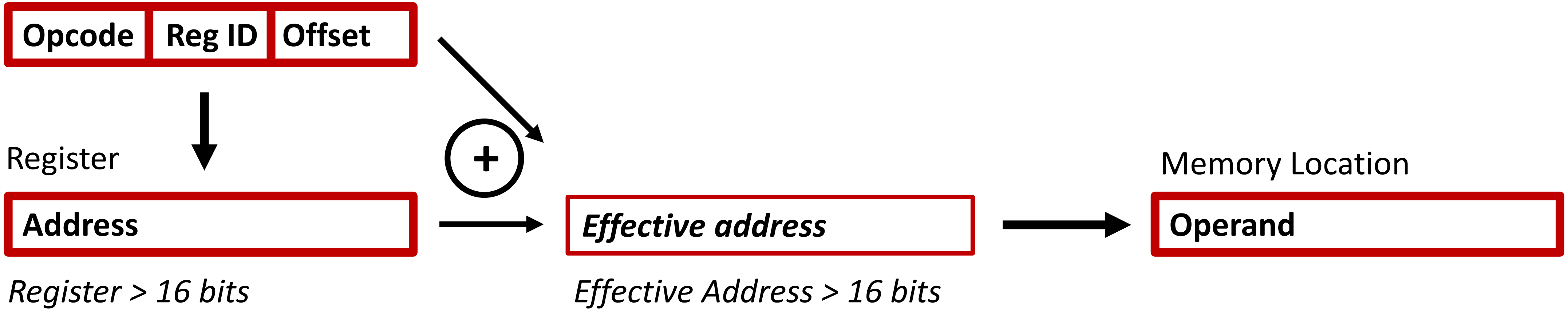
- Stack Addressing with Auto-increment

Addressing Limitations

- 16-bit addresses limit memory to a “flat” space of 65,536 (64K) words or bytes of memory.
- *Segmented* or *bank-switched* memory.
  - Memory is divided into 64K segments; a segment ID and an address are resolved to a physical (or virtual) memory location by the *Memory Management Unit* (MMU).

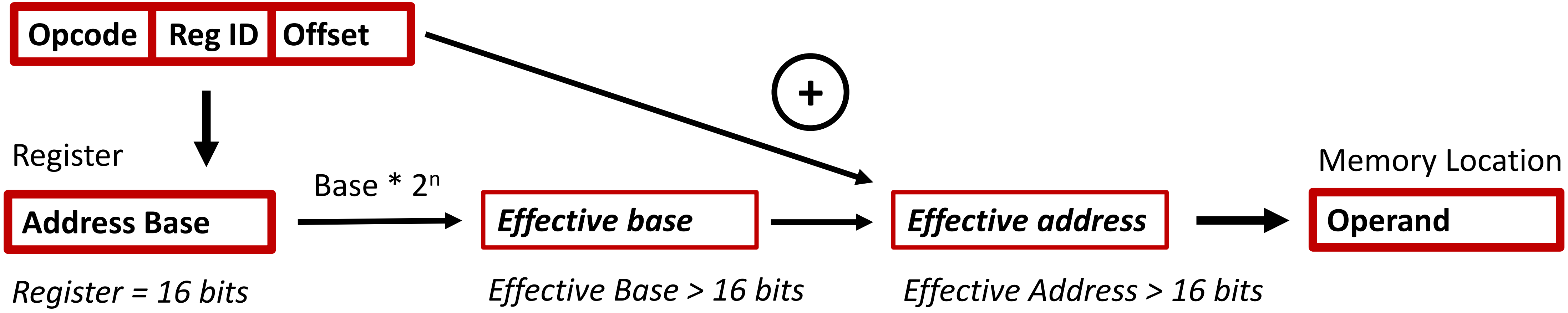
Addressing Limitations – Extended Base Register

- Displacement or Base Register



Addressing Limitations – Base Multiplier

- Displacement or Base Register



Impacts on Programming

- Multiple, complex addressing modes and memory management have made assembly language programming more difficult
- Compilers, linkers and loaders need to be *much* smarter than their early counterparts

The Natural Evolution of Processors

- Memory was *very* limited - ~ 1,000 words or less
- Most programmers were writing in machine code or assembly language
- By adding instructions that did more, programs required fewer instructions, and programmers didn't have to write as much code
- This led to what we now call Complex Instruction Set Computers (CISC)
- Over time, instruction sets grew more and more complex.
    - Harder for hardware designers to implement
    - Slower (more "cycles" required for complex instructions)

Microprogrammed Instruction Sets

- Complex instruction sets are very complex and hard to implement with logic design.
- *Microprogramming* is a way of implementing instruction sets by writing lower level instructions for a microcontroller… a processor inside the processor.
- Instructions are written in a *microassembler* language.
- Multiple microinstructions to implement one machine instruction
- Microinstructions are very low level: basic arithmetic and logic operations; register, memory, and bus access.

Microprogramming History

- The first microprogrammed machine was the EDSAC 2, developed by Maurice Wilkes at Cambridge in 1958.
  - The instruction sets at the time did not require microprogramming
  - There was not practical and cost-effective way to build a persistent microcontrol store
- The first widely successful microprogrammed computer was the IBM System/360, released in 1965.
- Microprograms may be stored in ROM, or in EPROM to allow instruction set updates.

The Rise of RISC

In 1980, *David Patterson* at UC Berkeley outlined an architecture for a *Reduced Instruction Set Computer*, and coined the term *RISC*.

- Fellow of the ACM
- Fellow of the IEEE
- Fellow of the Computer History Museum
- ACM Distinguished Service Award
- ACM-IEEE Eckert-Mauchly Award

RISC versus CISC

Reduced Instruction Set Computer

- One clock-cycle per instruction
- Effective *pipelining*
- Fewer addressing modes
- Requires more instructions per program (more RAM)
- Lower gate count
- Lower energy use

Complex Instruction Set Computer

- Multiple / variable clock-cycles per instruction
- More addressing modes
- Requires fewer instructions per program (less RAM)
- Higher gate count – more chip real estate
- Higher energy use

RISC Versus CISC Today

- No longer one versus the other
- RISC and CISC architectures have borrowed from one another
- Both are used

RISC

- MIPS, PowerPC, Atmel's AVR, the Microchip PIC processors, Arm processors, RISC-V
- Often used in mobile devices and for embedded applications

CSIC

- Motorola 68000 (68K), the DEC VAX, PDP-11, Intel x86

Instruction Pipelining (RISC computers)

- Each instruction is divided into several steps or *stages* – allowing instruction execution to be overlapped.
  Example: 5-stage pipeline

Clock Cycle	Inst 1	Inst 2	Inst 3	Inst 4	Inst 5	Inst 6	Inst 7
1	Fetch						
2	Decode	Fetch					
3	Execute	Decode	Fetch				
4	Memory Write	Execute	Decode	Fetch			
5	Register Write	Memory Write	Execute	Decode	Fetch		
6		Register Write	Memory Write	Execute	Decode	Fetch	
7			Register Write	Memory Write	Execute	Decode	Fetch

Instruction Pipelining - continued

Problems

- Branch instructions break the pipeline, and it must be reloaded
- Pipelines can be broken by *interrupts*
- Data dependencies occur when an instruction relies on the results of a previous instruction, causing the pipeline to *block*

Advances

- More stages… faster clock time, shorter blocks
- Parallel pipelines for conditional branch instructions

New Directions

- **Heterogeneous computing:** including multiple different computing elements (Application Specific Integrated Circuits, or *ASICs*) in a single system.
  - Graphics Processing Unit (GPU): common today
  - Machine learning
  - Image Processing
  - Cryptography
  - Video compression/decompression
  - Field Programmable Gate Arrays (FPGAs)

New Directions - continued

- **Near Memory Computing:** reducing the need to time-consuming fetch and store operations
- Rather than fetching small bits of data from memory to bring to the processor for computations, researchers are flipping this idea around. They are experimenting with building small processors directly into the memory controllers on your RAM or SSD.
- By doing the computation closer to the memory, there is the potential for huge energy and time savings since data doesn't need to be transferred around as much.
- *This idea is still in its infancy, but the results look promising.*

Quantum Computing

- Commercial computing started with Binary Coded Decimal arithmetic.
- Binary arithmetic allowed problems to be solved in *different* ways. It is faster the BCD, but has some problems.
- Quantum computing promises a significant new paradigm for computation.

Quantum Computing - continued

- Quantum computing is based on QUBITS – elements that can hold the value 0, 1, or some combination. This is called *superposition*.
- Another quantum property is *entanglement*, in which the value of one element is tied to another.
- Finally, quantum *interference* allows elements to affect the value of other elements – either positively or negatively. This leads to "voting" type solutions.

Quantum Computing - continued

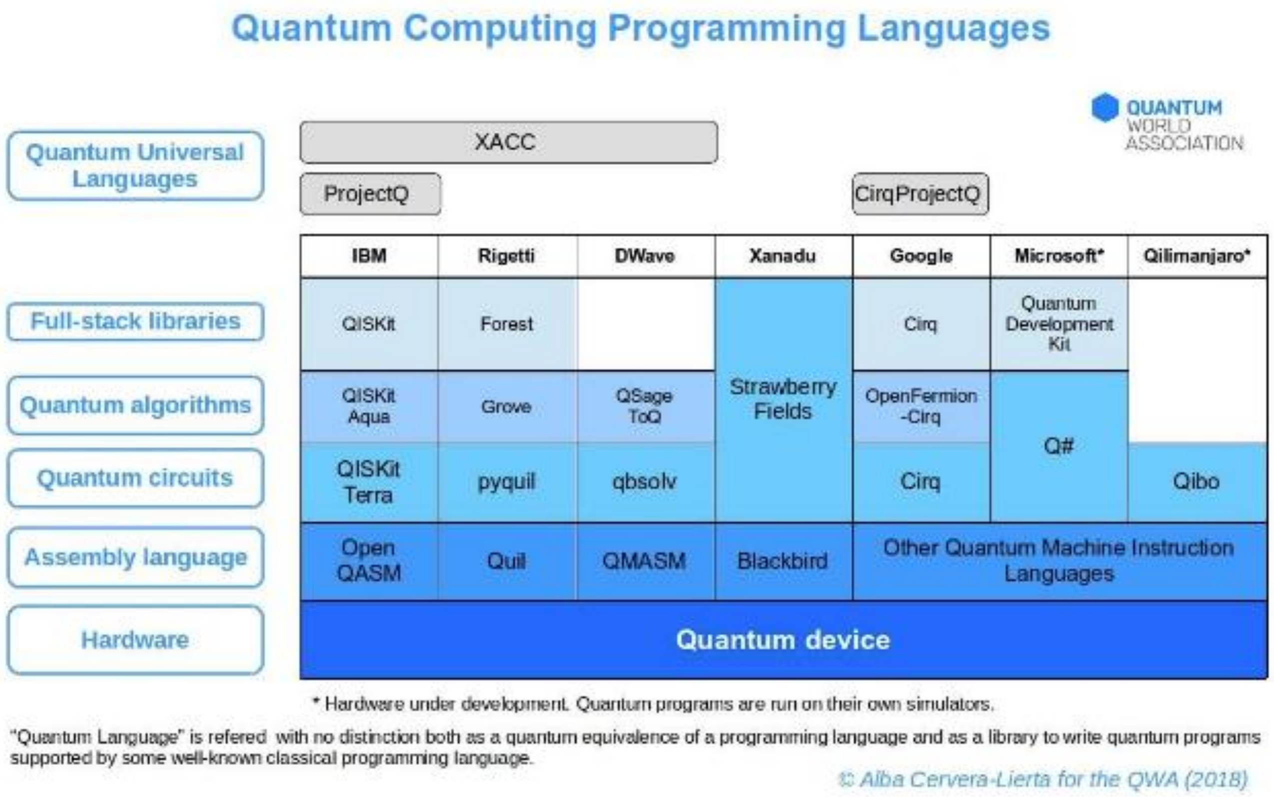
- Quantum computing offers the promise of simplifying certain classes of problems, including:
  - Modelling of natural systems
  - Searching
  - Machine learning
  - Artificial intelligence
- It would be a mistake to say quantum computers are *faster* than conventional computers. Rather, they will solve certain types of problems much faster – and possibly solve problems that can't be solved at all today.

Quantum Computing Today

- The largest quantum computers are about 50 Qubits.
- Two companies – Microsoft and IBM – have quantum simulators and development toolkits available.
- Current computers are subject to *quantum decoherence*, which causes loss of state information.
  - Quantum computing algorithms need to be fault-tolerant

Quantum Computing Today - continued

- New programming languages and libraries are being developed, but these languages are not yet standard. We are once again faced with *portability* issues.



Next Lecture

- Simplified Instructional Computer
  - A “typical” computer instruction set