Computer Organization

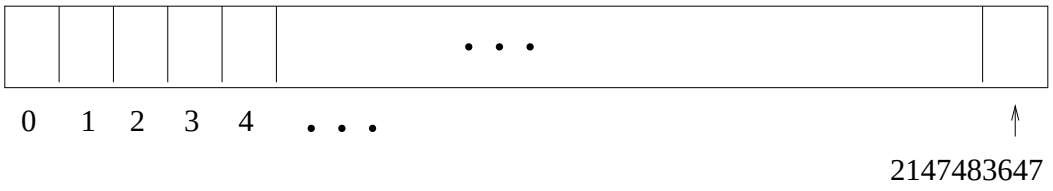
Memory

- The smallest possible quantity of information is a **binary digit**, often called a **bit**. In base ten, we have the digits 0, 1, 2, ..., 9. In base two, we have only the digits 0 and 1.
- A bit can only represent (at most) an answer to a yes/no question.
- It is relatively straightforward to represent the digits 0 or 1 as:
  - the state of an electronic circuit (memory),
    - current in a semiconductor device.
    - charge on a capacitor
  - or as an impression on a magnetizable material (hard drive, data tape)
  - or as a pit etched into an aluminum substrate (CDROM, DVD)
- Bits are grouped into units of eight bits, known as a byte. Why 8 ? It is a convenient size for representing a single letter in a character set.
- Q: How many different bit-patterns are possible in one byte ?
  A: $2^8 = 256$.

Computer memory can be thought of simply as a "parking lot". Each space in the memory can store one byte. The spaces are numbered, starting with 0, 1, 2, 3, ...

Our current model thinkpad has $2^{31} = 2,147,483,648$ bytes of memory (a.k.a. two gigabytes).

Pictorially,



Hardware Supported Data Types

ASCII Character Set

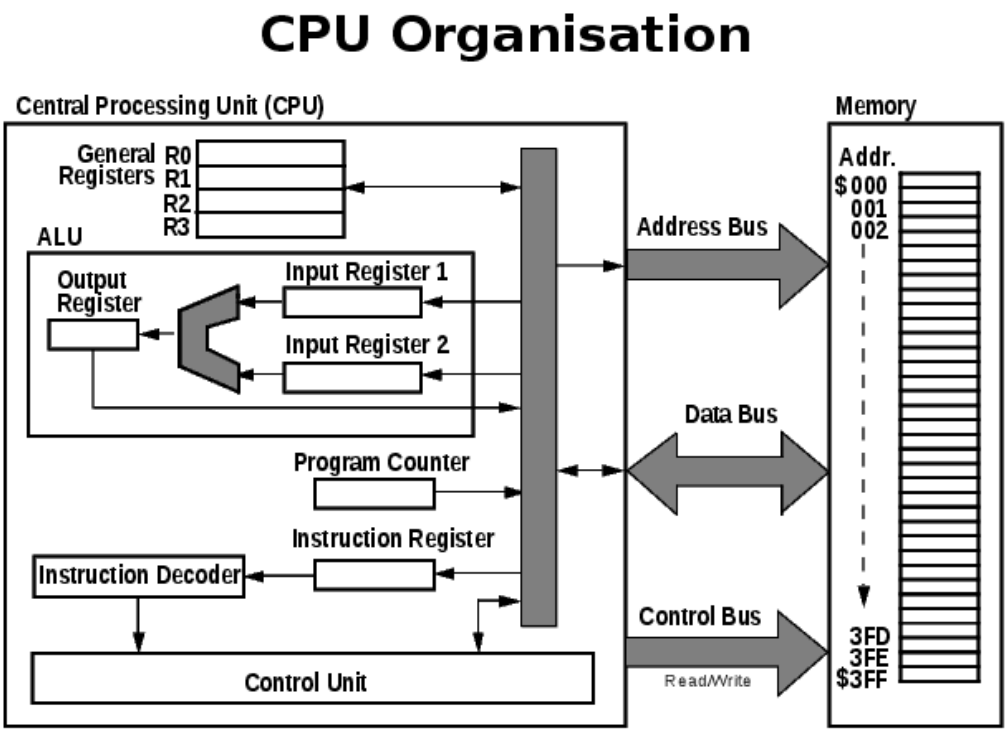
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of 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

- All integers are represented in base two.
  - Integers are usually 4 bytes; 2 byte integers are also supported on most computer chip-sets.
  - Long integers are 8 bytes.
  - Unsigned (positive) integers are usually either 4 or 8 bytes.
- Real numbers are represented by a **mantissa** and an **exponent**.
  - Floating point (real, e.g., 3.1415927) numbers are usually 4 bytes.
  - Double precision (real) numbers are usually 8 bytes.
  - Quad precision (real) numbers are usually 16 bytes.[1]

[1] Not supported on all chip-sets.

Simplified Computer Architecture



Computer Operation

- A program consists of a sequence of machine instructions held in memory
- The **program counter** holds the address of the next instruction.
- Operation proceeds by a **Fetch-Decode-Execute** cycle.
  - Program counter contains the address of the next instruction
  - The control unit initiates a fetch (from memory) of the next instruction. Upon completion of the memory cycle, the next instruction is held in the **instruction register**. An instruction includes:
    - An operation code which specifies the type of operation (e.g., add, multiply, branch).
    - An encoding of the data source used by the operation
  - The decode unit translates the operation code portion of the instruction into control signals which enable the correct data pathways within the CPU.
  - Instruction execution begins:
    - In case of an arithmetic instruction, the data in the specified general purpose registers are routed to the arithmetic logic unit where the appropriate operation is performed. The result is stored (temporarily) in the output register.

The contents of the output register are then routed to a specified general purpose registers.
Example: `add R1, R2, R3`

* In case of a **LOAD** instruction, the desired address is routed to the **memory address register** (MAR).[2] The control unit initiates a fetch (from memory) of the data contained at the memory location indicated by the MAR. The desired data is received in the **memory data register** (MDR). The data in the MDR is routed to a designated general purpose register. The net result of the load instruction is that the data in a specified memory location is copied into a specified general purpose register.
Example: `load [ R1 ], R2`
* A **STORE** instruction works similarly to the LOAD instruction, except that the data in a specified general purpose register is copied into a specified memory location.
Example: `store R0, [ R3 ]`
* A **COMPARE** instruction compares (subtracts) two numbers and sets the **condition code register** (CC). The condition code register indicates exactly one of "less", "equal", or "greater".
Example: `CMP R1, R2`
* An **UNCONDITIONAL BRANCH** instruction changes the program counter to a specified instruction (possibly anywhere) within the currently executing program.
Example: `BR` *address*
* A **CONDITIONAL BRANCH** instruction comes in several variations, such as "branch if less", "branch if less or equal", etc. The conditional branch instruction uses the condition code register and takes the branch (i.e., sets the PC) if the condition code register matches the requirements of the particular conditional branch instruction.
Example: `BE` *address*
The above branch is only taken if the condition code indicates "equals".
* A **CALL** instruction is a special type of unconditional branch instruction which stores the current value of the **program counter** (so that execution can later resume at the subsequent instruction), and then changes the program counter to a specified instruction (possibly anywhere) within the program.
Example: `CALL` *address*
* A **RETURN** instruction is a special type of unconditional branch instruction which uses a previously stored value of the program counter (saved by the CALL) to resume execution at the point immediately following the CALL.
* An **INPUT** or an **OUTPUT** instruction utilizes hardware not shown in the diagram. For our purposes, we can consider an INPUT instruction as a data copy from an input device to a (reserved) location in memory. We can consider an OUTPUT instruction as a data copy from a (reserved) location in memory to an output device.

For all instructions, except the branch instructions, the program counter is advanced to the

[2] The MAR and the MDR are not shown in the diagram above.

next instruction in the program. I.e., if instructions take up 4 bytes, then the number 4 is simply added to the program counter.

Summary of Computer Instructions

A computer program can:

- perform an arithmetic operation,
- copy a value from a place in memory to a register,
- copy a value in a register to a place in memory,
- unconditionally go to a new position within the program,
- compare two values, and save the result in the condition code register,
- conditionally go to a new position within the program depending on the condition code register,
- call a sub-program,
- return from a sub-program,
- input a value,
- or, output a value.

That's it.

Algorithms

An **algorithm** is a sequence of steps which can be performed by a computer.

Every algorithm must either be:

1. expressed entirely using instructions which a computer can perform **OR**
2. expressed in sufficient detail such that the language describing the algorithm can be directly translated in to a sequence of instructions.

Addressing Modes

There are many variations on addressing modes, and we will examine them in greater detail later in the semester. For now, we introduce only a few addressing modes:

Register direct: For example, and ADD instruction might be:

```
add   R1, R2, R3
```

In this addressing mode, a register holding the data to be used is specified. For the example above, if R1 contains 7, and R2 contains 11, then on completion of the `add` instruction, R3 contains 18.

Immediate mode: Immediate mode allows small data values to be specified in the instruction itself. For example:

```
add   R1, 17, R3
```

In the example above, the value held in register R1 is added to the number 17, and the result is stored in R3.

Register indirect: In the register indirect addressing mode, a register is specified which holds the **location in memory** where the data may be found. This mode is usually distinguished from register direct mode by using square brackets around the register name. For example:

```
load   [R1], R2
```

In the example above, the data found in memory at the location contained in register R1 is copied to R2.

Some computer chip-sets allow register indirect mode in arithmetic operations (e.g., Intel x86 chips), others require register direct addressing for arithmetic operations (e.g., SPARC, MIPS).