

PeANUt Machine Language and Operation

- ref: [PeANUt Spec, sect 2, Appendix C]
additional reading: [O'H&Bryant, sect 3.1–3.5]
- PeANUt initialisation
- PeANUt machine language (initialization files)
- basic machine operation:
 - addition
 - logical and
 - subtraction

Initialising the PeANUt

- a microprocessor automatically executes a sequence of instructions
- for useful operation, we need to **initialise** a microprocessor, by defining its **starting state**
- the initialisation process: defines the **starting state**:
 - the state of the memory: to contain an image of executable program
 - the state of the PC (program counter): to the **program start address**
 - the state of other registers: to **0** (except SP)
- this can be done using a file specifying the desired state:

an `.mli` (Machine Language Initialisation) file

- create a file `myfile.mli`
- provide start PC value by a **START** `<address>` directive
- not every memory cell needs to be mentioned
- define memory by listing cell contents
- each such list has a starting point specified by a **AT** `<address>` directive

Machine Language Initialisation Files

- e.g. addition.mli:

```
START a10 ; start address of program, init. PC to this
AT a1 ; address where numbers and result is stored
000 000 000 000 010 0 ; a1: decimal 4
000 000 000 000 010 1 ; a2: decimal 5
000 000 000 000 000 0 ; a3: decimal 0 (result will be stored into
AT a10 ; store the following instrns from a10 onward
001 001 0 000 000 001 ; a10: load mem[a1]
001 011 0 000 000 010 ; a11: add mem[a2]
001 010 0 000 000 011 ; a12: store mem[a3]
110101 0 000 000 001 ; a13: trap 1 (halt)
```

- how is the PeANUt initialised?

- line of the form **START** aX , where X_8 specifies the initial program counter (PC) value
- lines of the form **AT** aX , where X_8 is the address of a block of memory cells into which the following (16-bit) data items go

More on `.mli` Files

- in creating the image for the PeANUt, some things are ignored
 - comments: anything written to the right of a `';`
 - blank lines
 - spaces and tabs in data values(essential for human readability though!)
- restrictions:
 - there can only be one **START** directive
 - there may be many **AT** directives, but:
 - ◆ they must not result in overlapping data
 - ◆ they must appear in sequence
- `.mli` files are (trivially) converted to `.img` (image) files (binary files)
 - image files can directly initialise the PeANUt; `.mli` files cannot!

An Example: printword.mli

```
; Simple example machine language program to print a word
```

```
START a10 ; start address of program, init. PC to this
```

```
AT a10 ; store the following data items (instrns)
```

```
; into memory from a10 onwards
```

```
000 001 0 001 001 000 ; a10: load 'H' (immediate mode)
```

```
110101 0 000 000 011 ; a11: trap 3 (put)
```

```
000 001 0 001 000 101 ; a12: load 'E' (immediate mode)
```

```
110101 0 000 000 011 ; a13: trap 3 (put)
```

```
000 001 0 001 001 100 ; a14: load 'L' (immediate mode)
```

```
110101 0 000 000 011 ; a15: trap 3 (put)
```

```
000 001 0 001 001 100 ; a16: load 'L' (immediate mode)
```

```
110101 0 000 000 011 ; a17: trap 3 (put)
```

```
000 001 0 001 001 111 ; a20: load '0' (immediate mode)
```

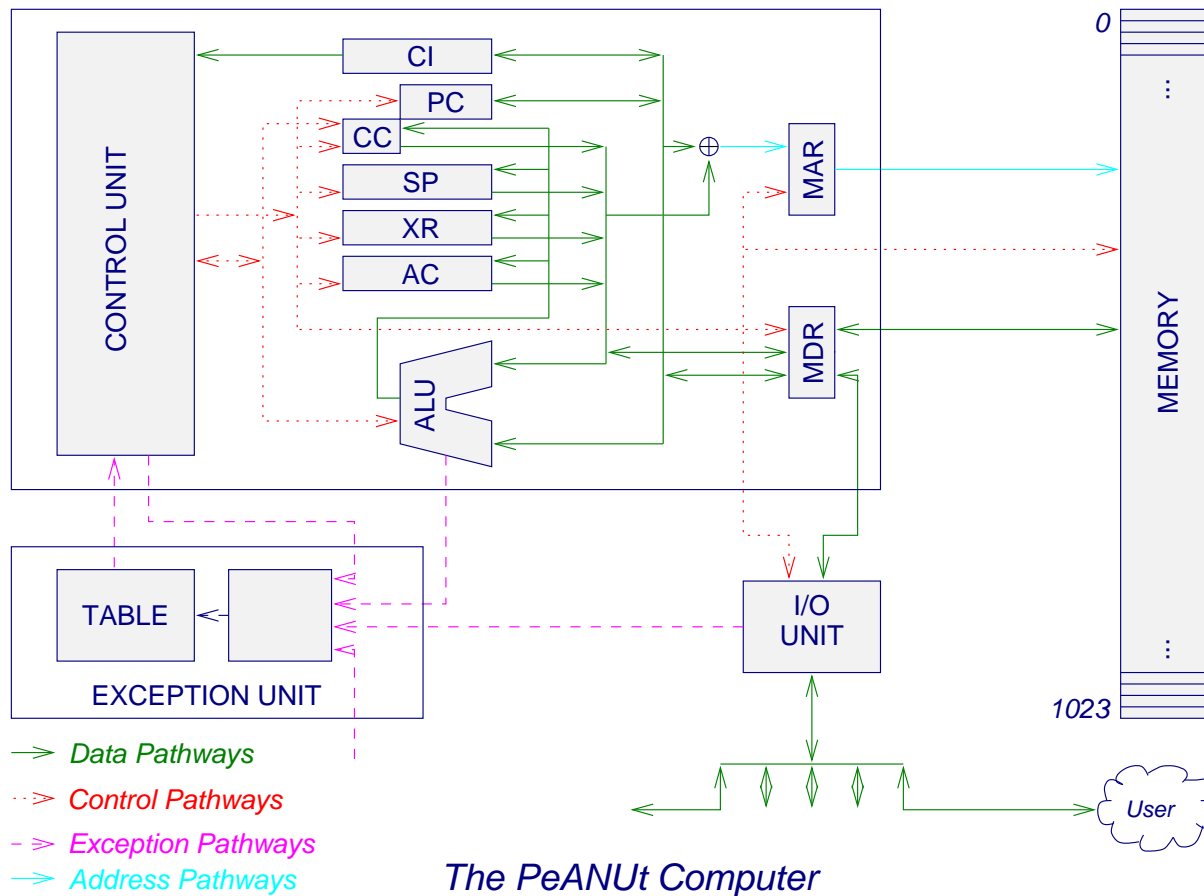
```
110101 0 000 000 011 ; a21: trap 3 (put)
```

```
000 001 0 000 001 010 ; a22: load '\n' (new line) (imm. mode)
```

```
110101 0 000 000 011 ; a23: trap 3 (put)
```

```
110101 0 000 000 001 ; a24: trap 1 (halt)
```


Addition (immediate mode) – Inside the Machine



- load 5:
 $AC \leftarrow CI[9..0]$
- add 14:
 $AC \leftarrow AC + CI[9..0]$
- store mem[a35]:
 $MAR \leftarrow CI[9..0]$
 $MDR \leftarrow AC$
Write, Enable
 $mem[a35] \leftarrow MDR$

Addition (direct mode)

- $\text{mem}[a3] \leftarrow \text{mem}[a1] + \text{mem}[a2]$

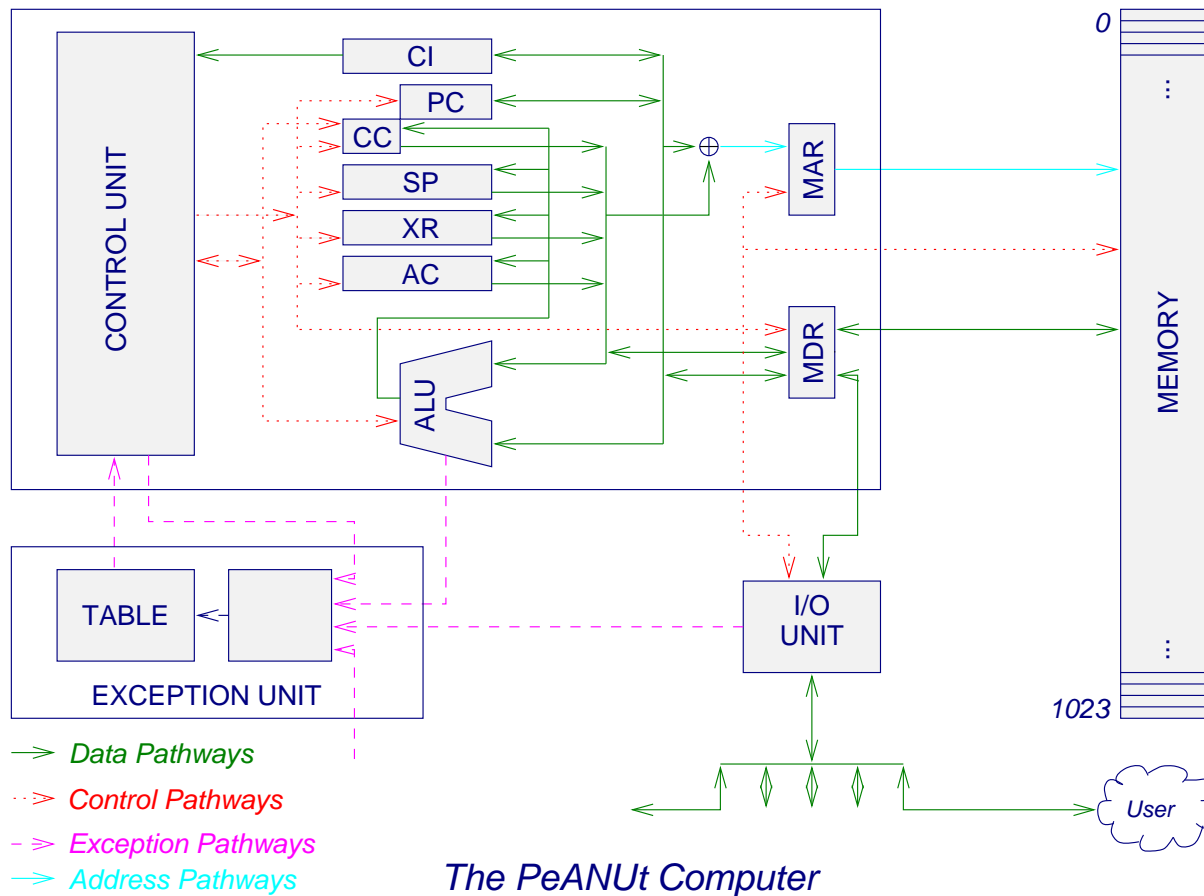
	instruction	AC	mem[a3]
(say $\text{mem}[a1] = 4$, $\text{mem}[a2] = 5$)	load mem[a1]	4	?
	add mem[a2]	9	?
	store mem[a3]	9	9

- machine code

(need to include **AT** and **START**)

001	001	0	000	000	001	; load mem[a1]
001	011	0	000	000	010	; add mem[a2]
001	010	0	000	000	011	; store mem[a3]
110101		0	000	000	001	; halt (trap 1)

Addition (direct mode) – Inside the Machine



- load mem[a1]:
 - MAR ← CI[9..0]
 - Read, Enable
 - MDR ← mem[a1]
 - AC ← MDR
- add mem[a2]:
 - MAR ← CI[9..0]
 - Read, Enable
 - MDR ← mem[a2]
 - AC ← AC + MDR
- store mem[a3]:
 - MAR ← CI[9..0]
 - MDR ← AC
 - Write, Enable
 - mem[a3] ← MDR

Logical And (direct mode)

- $\text{mem}[a1] \leftarrow \text{mem}[a63] \text{ AND } \text{mem}[a17]$

assume:

$\text{mem}[a63] = 000\ 000\ 00000\ 11100$

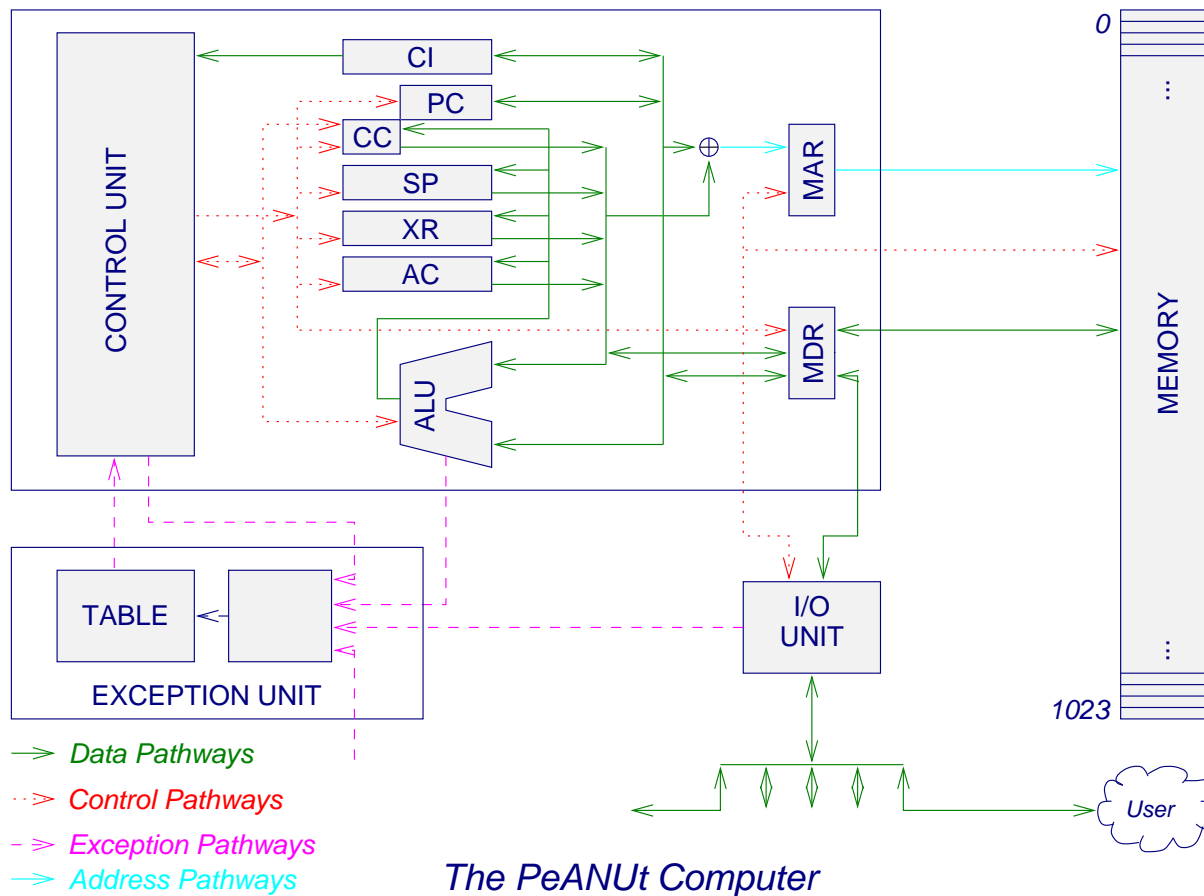
$\text{mem}[a17] = 101\ 010\ 10101\ 01010$

instruction	AC	mem[a1]
load mem[a63]	..011100	?
and mem[a17]	..001000	?
store mem[a1]	..001000	..001000

- machine code: (need to include **AT** and **START** to make it valid)

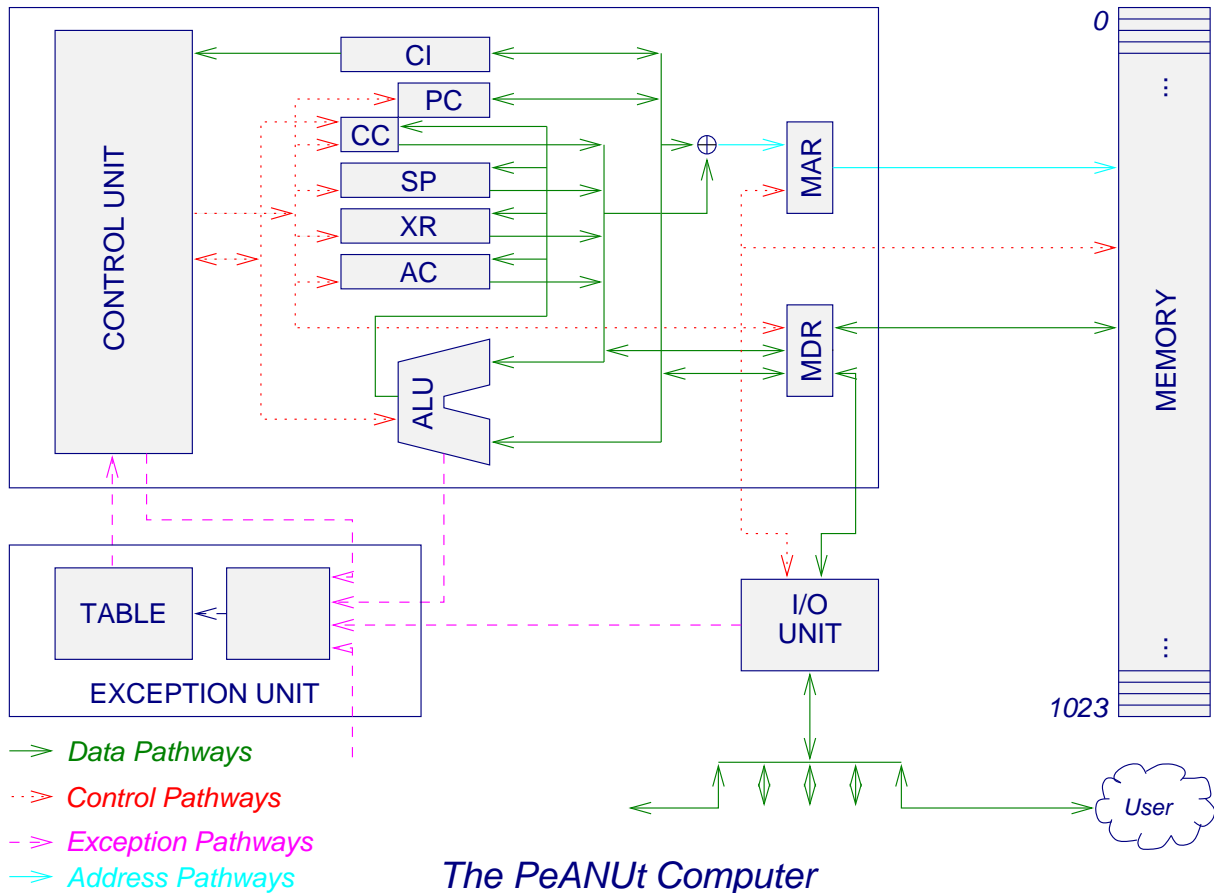
```
001 001 0 000 110 011 ; load mem[a63]
101110 0 000 001 111 ; and mem[a17]
001 010 0 000 000 001 ; store mem[a1]
110101 0 000 000 001 ; halt (trap 1)
```

Logical And (direct mode) – Inside the machine



- **load mem[a63]:**
 - MAR ← CI[9..0]
 - Read, Enable
 - MDR ← mem[a63]
 - AC ← MDR
- **and mem[a17]:**
 - MAR ← CI[9..0]
 - Read, Enable
 - MDR ← mem[a17]
 - AC ← AC AND MDR
 - MDR
- **store mem[a1]:**
 - MAR ← CI[9..0]
 - MDR ← AC
 - Write, Enable
 - mem[a1] ← MDR

Appendix: Subtraction (indirect mode) – Inside the Machine



● load 20:

$AC \leftarrow CI[9..0]$

● sub mem[mem[a10]]:

$MAR \leftarrow CI[9..0]$

Read, Enable

$MDR \leftarrow mem[a10]$

$MAR \leftarrow MDR[9..0]$

Read, Enable

$MDR \leftarrow$

$mem[mem[a10]]$

($mem[a52]$)

$AC \leftarrow AC - MDR$

● store mem[a23]:

$MAR \leftarrow CI[9..0]$

$MDR \leftarrow AC$

Write, Enable

$mem[a23] \leftarrow MDR$