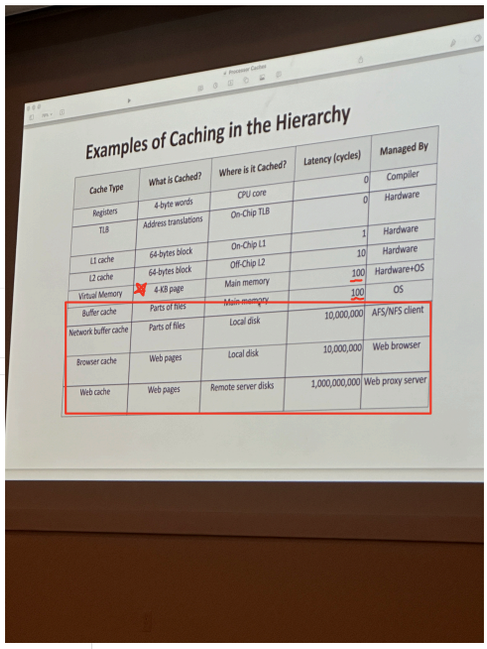
SLS Lecture 16 : Caches : Constructing them from first principles

Contents




```
%run -i ../python/common.py
UC_SKIPTERMS=True
%run -i ../python/ln_preamble.py
```

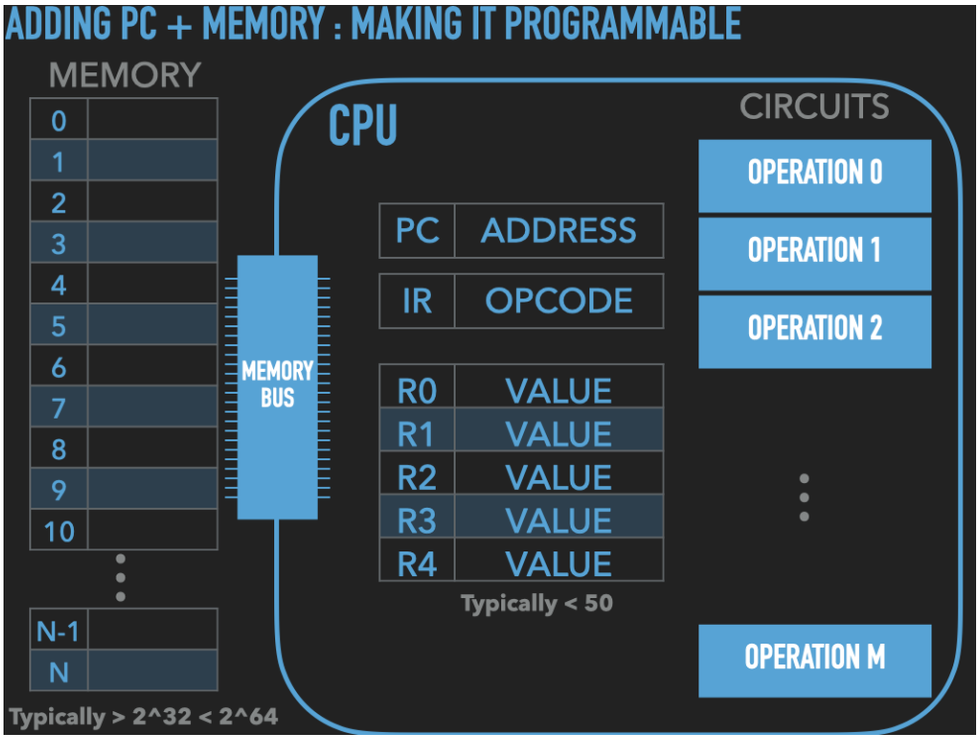
```
# setup for sumit examples
appdir=os.getenv('HOME')
appdir=appdir + "/caches"
#print(movdir)
output=runTermCmd("[[ -d " + appdir + " ]] &&  rm -rf "+ appdir +
            ";mkdir " + appdir +
            ";cp ../src/SOL6502/Makefile ../src/SOL6502/sum.s ../src/SOL6502/sum.txt
../src/SOL6502/SOL6502.cfg " + appdir)

display(Markdown('''
- create a directory `mkdir cache; cd cache`
- copy examples
- add a `Makefile` to automate assembling and linking
    - we are going run the commands by hand this time to highlight the details
'''))
TermShellCmd("ls " + appdir)
```

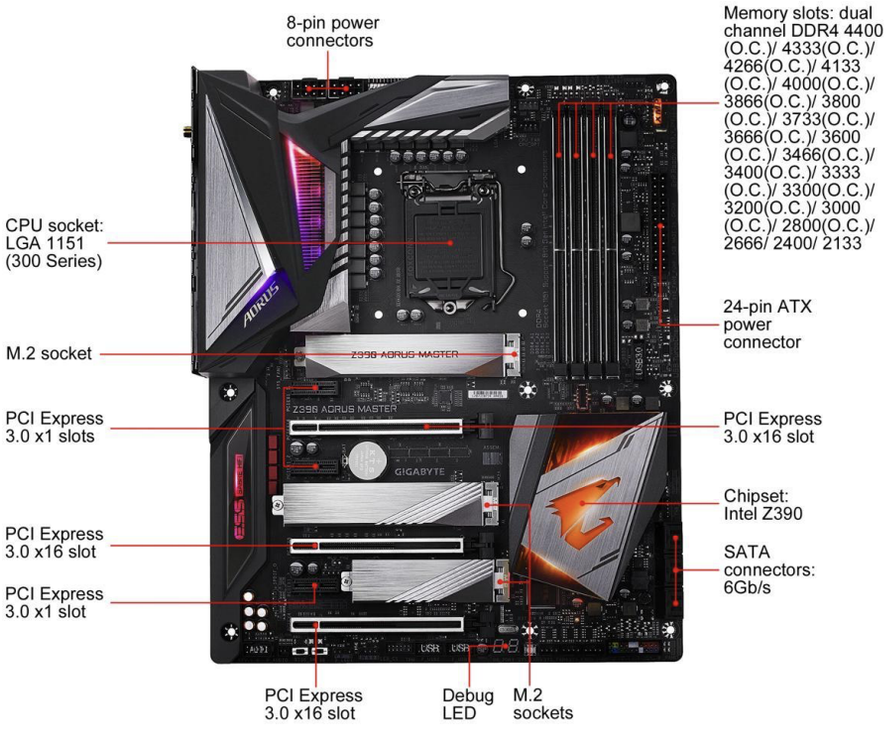
- create a directory `mkdir cache; cd cache`
- copy examples
- add a `Makefile` to automate assembling and linking
    - we are going run the commands by hand this time to highlight the details

```
$ ls /home/jovyan/caches
Makefile  SOL6502.cfg  sum.s  sum.txt
$
```

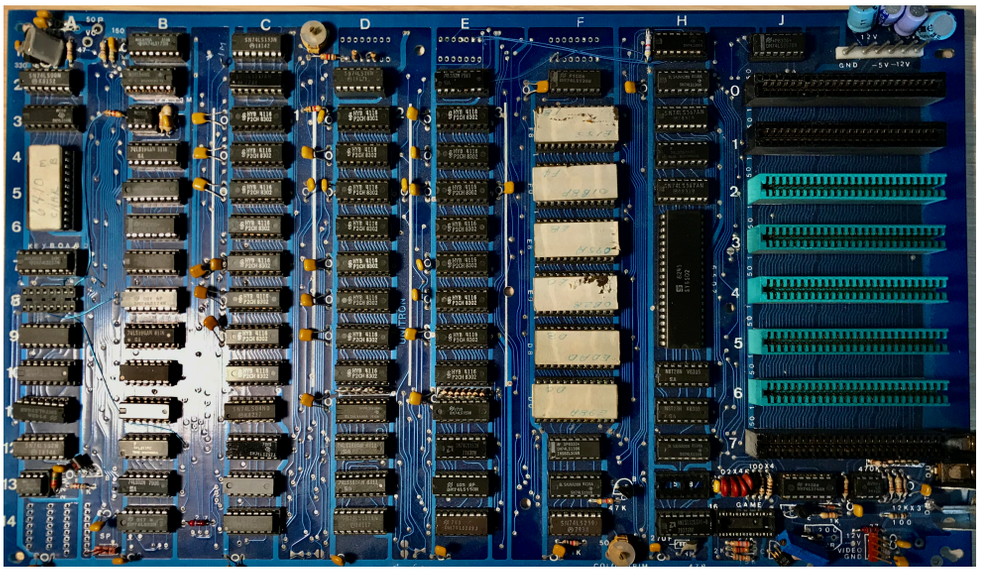
16.1. Remember this picture



16.1.1. What is the physical Reality



16.2. Lets build our Computer



16.3. A nice Simple CPU - Only 6 registers and no MMU

MOS 6502 registers

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	*(bit position, hex)*
Main registers																
								A								**A**ccumulator
Index registers																
								X								**X** index
								Y								**Y** index
0	0	0	0	0	0	0	1	S								**S**tack **P**ointer
Program counter																
PC																**P**rogram **C**ounter
Status register																
								N	V	-	B	D	I	Z	C	**P** Processor flags

16.4. Simple Main Memory

- Physical Memory : $2^{16} = 65536 \text{ Bytes} = 64 \text{ KiloBytes (Kb)}$
- NO Virtual Memory!

16.5. The Code

Code we have come to love: A 6502 version of sumit!

```
display(Markdown(FileCodeBox(
    file="../src/SOL6502/sum.s",
    lang="gas",
    title="<b>CODE: 6502 asm - sum.s",
    h="100%",
    w="107em"
)))
```

CODE: 6502 asm - sum.s

```
        .org $0000     ; starting at address 0x0000
        ;; We place things in memory locations by hand
        ;; Fill memory 0x0000 - 0xE000 with zeros
        .repeat $E000
        .byte $00
        .endrep

        ;; Put our data at 0xE000
        .byte 10                ; 0xE000 Array length
        .byte 1                 ; Array[0] = 1
        .byte 2                 ; Array[1] = 2
        .byte 3                 ; Array[2] = 3
        .byte 4                 ; Array[3] = 4
        .byte 5                 ; Array[4] = 5
        .byte 6                 ; Array[5] = 6
        .byte 7                 ; Array[6] = 7
        .byte 8                 ; Array[7] = 8
        .byte 9                 ; Array[8] = 9
        .byte 10                ; Array[9] = 10

        ;; Fill memory from end of data to 0xF000 with zero
        .repeat $1000-11
        .byte $00
        .endrep

        ;; Put our code at 0xF000
        ;; Set address to F000 (this is where our code will live)
        .ORG $F000
        LDA #0          ; load A register with 0
        LDX #0          ; load X register with 0
LOOP:
        CPX $E000       ; compare value in X register with value at E000 (length of Array)
        BEQ DONE        ; if equal then jump to done
        ADC $E001,X     ; add value in memory  at M[0xE001 + X register] : A = A + Array[X]
        INX             ; X=X+1
        JMP LOOP
DONE:
        BRK
```

16.6. Our old friends : Assembler and linker

- ca65 - assembler different syntax but same idea
- ld65 - linker but only using for symbol resolution we are taking care of placing things in memory

```
TermShellCmd("make sum.img", cwd=appdir, prompt='')
```

```
ca65 sum.s -l sum.o.lst -o sum.o
ld65 -o sum.img -C SOL6502.cfg sum.o
rm sum.o
```

```
display(Markdown(FileCodeBox(
    file= appdir + "/sum.o.lst",
    lang="gas",
    title="<b>CODE: 6502 asm listing file",
    h="100%",
    w="107em"
)))
```

CODE: 6502 asm listing file

```
ca65 V2.18 - Ubuntu 2.18-1
Main file   : sum.s
Current file: sum.s

000000r 1                       .org $0000     ; starting at address 0x0000
000000  1                       ;; We place things in memory locations by hand
000000  1                       ;; Fill memory 0x0000 - 0xE000 with zeros
000000  1  00 00 00 00          .repeat $E000
000004  1  00 00 00 00
000008  1  00 00 00 00
00E000  1                       .byte $00
00E000  1                       .endrep
00E000  1
00E000  1                       ;; Put our data at 0xE000
00E000  1  0A                   .byte 10                ; 0xE000 Array length
00E001  1  01                   .byte 1                 ; Array[0] = 1
00E002  1  02                   .byte 2                 ; Array[1] = 2
00E003  1  03                   .byte 3                 ; Array[2] = 3
00E004  1  04                   .byte 4                 ; Array[3] = 4
00E005  1  05                   .byte 5                 ; Array[4] = 5
00E006  1  06                   .byte 6                 ; Array[5] = 6
00E007  1  07                   .byte 7                 ; Array[6] = 7
00E008  1  08                   .byte 8                 ; Array[7] = 8
00E009  1  09                   .byte 9                 ; Array[8] = 9
00E00A  1  0A                   .byte 10                ; Array[9] = 10
00E00B  1
00E00B  1                       ;; Fill memory from end of data to 0xF000 with zero
00E00B  1  00 00 00 00          .repeat $1000-11
00E00F  1  00 00 00 00
00E013  1  00 00 00 00
00F000  1                       .byte $00
00F000  1                       .endrep
00F000  1
00F000  1                       ;; Put our code at 0xF000
00F000  1                       ;; Set address to F000 (this is where our code will live)
00F000  1                       .ORG $F000
00F000  1  A9 00                LDA #0          ; load A register with 0
00F002  1  A2 00                LDX #0          ; load X register with 0
00F004  1                 LOOP:
00F004  1  EC 00 E0             CPX $E000       ; compare value in X register with value at E000 (length of Array)
00F007  1  F0 07                BEQ DONE        ; if equal then jump to done
00F009  1  7D 01 E0             ADC $E001,X     ; add value in memory  at M[0xE001 + X register] : A = A + Array[X]
00F00C  1  E8                   INX             ; X=X+1
00F00D  1  4C 04 F0             JMP LOOP
00F010  1                 DONE:
00F010  1  00                   BRK
00F010  1
```

16.6.1. The “binary”: A Simple Image file

The linker produce a simple binary image file that is an exact copy of memory to load

```
TermShellCmd("od -Ax -t x1  sum.img", cwd=appdir, prompt='$ ')
```

```
$ od -Ax -t x1  sum.img
000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
00e000 0a 01 02 03 04 05 06 07 08 09 0a 00 00 00 00 00
00e010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
00f000 a9 00 a2 00 ec 00 e0 f0 07 7d 01 e0 e8 4c 04 f0
00f010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
010000
$
```

16.6.2. Ok Now What?

```
display(Markdown(FileCodeBox(
    file="../src/SOL6502/sum.txt",
    lang="gas",
    title="<b>CODE: 6502 Loop",
    h="100%",
    w="107em"
)))
```

CODE: 6502 Loop

```
Fetch:
a) Buses : Read Addr = PC -> Value
           IR = Value

Decode:
a) lookup IR and tell execute what to do
A9 : LDA IMM
A2 : LDX
EC : CPX ABS
F0 : BEQ PC Rel
7D : ADC ABS
E8 : INX
4C : JMP ABS
00 : BRK




Execute
LDA IMM :
a) Buses : Read Addr = PC +1 -> Value
b) A = Value
c) PC = PC + 2

LDX :
a) Buses : Read Addr = PC +1 -> Value
b) X = Value
c) PC = PC + 2

CPX ABS :
a) Buses : Read Addr = PC + 1 -> Value
b) TempAddr Low Byte = Value
c) Buses : Read Addr = PC + 2 -> Value
d) TempAddr High Byte = Value
e) Buses : Read Addr = TempAddr -> Value
f) Compare : TempVal = X - Value
g) Set P flags : if TempVal == 0 then P.Z = 1 else P.Z = 0
h) PC = PC + 3

BEQ PC Rel :
a) if P.Z == 1 then
   b) Buses : Read Addr = PC + 1 -> Value
   c) PC = PC + Value
d) else
   e) PC = PC + 2

ADC ABS :
a) Buses : Read Addr = PC + 1 -> Value
b) TempAddr Low Byte = Value
c) Buses : Read Addr = PC + 2 -> Value
d) TempAddr High Byte = Value
e) Buses : Read Addr = TempAddr + X -> Value
f) Add : A = A + Value
g) PC = PC + 3

INX :
a) X = X + 1
b) PC = PC + 1

JMP ABS :
a) Buses : Read Addr = PC + 1 -> Value
b) TempAddr Low Byte = Value
c) Buses : Read Addr = PC + 2 -> Value
d) TempAddr High Byte = Value
e) PC = TempAddr

BRK:
a) STOP!!!
```

16.7. Processor Caches

Modern CPU's are very fast and memory is "far away" and relatively slow.

- **Notice that a lot of what a program is accessing memory**
- **Since memory is "slow" most of our CPU time is spent "IDLE" waiting for memory!**
- **Caches are critical to achieving high performance on a modern CPU**

https://ark.intel.com/content/www/us/en/ark/products/203908/intel-core-i710700e-processor-16m-cache-up-to-4-50-ghz.html

Miss Rate

- Fraction of memory references not found in cache (misses / accesses) = 1 - hit rate
- Typical numbers
- 3-10%

Hit time

miss penalty

Let's think about those numbers

Huge difference between a hit and a miss

Could be 100x, if just L1 and main memory

Would you believe 99% hits is twice as good as 99%?

Consider:

Cache hit : 1 cycle

Cache miss :