

Örnek ① word add.

lw s3, 1(zero) } read memory word 1 into s3

sw t4, 0x3(zero) } write the value in t4 to memory word 3

örnek ① byte add.

lw s3, 8(zero) } read word at add. 8 into s3

sw t7, 0x10(zero) } write t7 into add. 16 (word 4)
↳ hex.

örnek ① constant

int a = -372;

int b = a + 6;

// int is a 32-bit signed word

addi s0, zero, -372

addi s1, s0, 6

#s0 = a s1 = b

örnek ② constant

int a = 0xFEDC8765

lui s0, 0xFEDC8

addi s0, s0, 0x765

örnek ① addi / lui

int a = 0xFEDC8EAB

#s0 = a

lui s0, 0xFEDC8 #s0 = 0xFEDC9000

addi s0, s0, 0xEAB #s0 = 0xFEDC9000

+ 0xFFFFEAB

0xFEDC8EAB

branching örnek ①

addi s0, zero, 4	#s0 = 0 + 4 = 4
addi s1, zero, 1	#s1 = 0 + 1 = 1
slli s1, s1, 2	#s1'i 4'e carp $\Rightarrow 4 = s1$
beq s0, s1, target	#s0 = s1 ise target
addi s1, s1, 1	} adisuz
sub s1, s1, s0	

target:

add s1, s1, s0

} s1 = s1 + s0 = 8

örnek ② branching

j target

srai s1, s1, 2

addi s1, s1, 1

sub s1, s1, s0

#jump to target

} not executed

target:

add s1, s1, s0

} s1 = s1 + s0

Loops break ①

```
int sum=0;
```

```
int i;
```

```
for (#1; i < 101; i = i * 2) {
```

```
    sum = sum + i;
```

```
}
```

```
#s0 = i, s1 = sum
```

```
addi s1, zero, 0
```

```
addi s0, zero, 1
```

```
addi t0, zero, 101
```

```
loop:
```

→ set less than
sli t2, s0, t0 #if $s0 < t0$, $t2 = 1$ else $t2 = 0$

```
beq t2, zero, done
```

```
add s1, s1, s0
```

```
slli s0, s0, 1
```

```
jne loop
```

```
done:
```


Function calls ornek ①

```
int main() {  
    simple();  
    a = b + c;  
}
```

0x000000300 main: jal simple
0x000000304 add s0, s1, s2

→ call

```
void simple() {  
    return;  
}
```

0x00000051C simple: jr ra

→ return

jal simple:

ra = PC + 4 (0x000000304)

jumps to simple label (PC = 0x00000051C)

jr ra:

PC = ra (0x000000304)

Function calls örnek ②

```
int main() {
```

```
    int y;
```

```
    ...
```

```
    y = diffOfSums(2, 3, 4, 5); // 4 arguments
```

```
    // 4 arguments
```

```
int diffOfSums(int f, int g, int h, int i) {
```

```
    int result;
```

```
    result = (f + g) - (h + i);
```

```
    return result;
```

```
}
```

```
#s7 = y
```

```
main:
```

```
...
```

```
addi a0, zero, 2
```

```
addi a1, zero, 3
```

```
addi a2, zero, 4
```

```
addi a3, zero, 5
```

```
jal diffOfSums
```

```
add s7, a0, zero
```

```
...
```

```
#s3 = result
```

```
diffOfSums:
```

```
add t0, a0, a1
```

```
add t1, a2, a3
```

```
sub s3, t0, t1
```

```
add a0, s3, zero
```

```
jr ra
```

NOT: diffOfSums overwrote
3 reg. (t0, t1, s3)

can use stack to
temporarily store
registers.

Örnek Directmapped-cache

1024 blok, 4KB cache, 32-bit add.

① 1024 adet blok var, 4KB toplam alan $4KB/1024 = 4 \text{ byte}$
 $= 32 \text{ bit}$

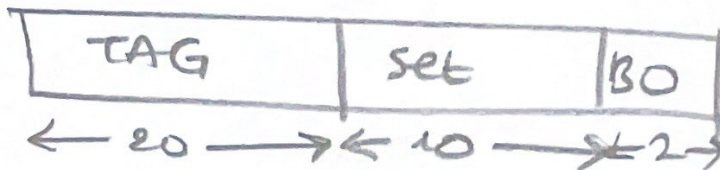
② Byte Offset = $\frac{\text{blok genişliği}}{1 \text{ word}} = 4 \text{ byte}$, 1 word'ün belirli bir konuma erişebilmek için var.

$$\log_2 4 = 2 \text{ bit}$$

③ Set = hangi satırda olduğuna bakmak için var.

$$\log_2 1024 = 10 \text{ bit}$$

④ 32-bit ten kalan TAG olur.



Örnek Direct-mapped

16KB, block size = 4 word, 64-bit add.

① 128 bit block size = 16 byte

$$\log_2 16 = 4 \rightarrow \text{BO}$$

② $\frac{16 \cdot 2^{10}}{4 \cdot 4} = 2^{10}$ satır sayısı

TAG	set	BO
50	10	4

$$\log_2 2^{10} = 10 \rightarrow \text{set}$$

③ $64 - 14 = \text{TAG} = 50$

Q2: 2000 mem. access, 1250 cache de var.
 $t_{\text{cache}} = 1$ cycle, $t_{\text{mem}} = 100$ cycles.
~~What~~ cache hit and miss rate?

AMAT?

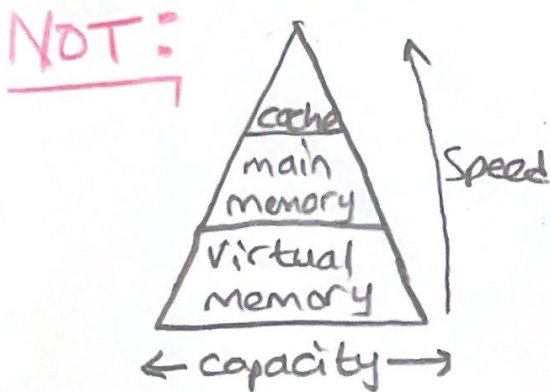
$$\frac{1250}{2000} = 0.625 \text{ hit rate} \quad 1 - 0.625 = 0.375 \text{ miss rate}$$

$$1 + 0.375(100 + 0) = 38.5 = \text{AMAT}$$

NOT: Multicycle 'da
 3 cycle beq
 4 " A-type, addi,
 swjal
 5 " lw

NOT: RF is faster than
 memory, writing
 memory is faster
 than reading me-
 mory

NOT: CPI = 4.12 cycle/ins.
 $T_{c-multi} = 375 \text{ ps}$
 $\text{Exe. time} = \#ins * CPI * T_c$



NOT: Temporal locality:
 Bir kez kullanılmışı tut
 Spatial " :
 " " kullanılmışın yan-
 nındaki lerini tut.

NOT: AMAT:
 $t_{cache} + MR_{cache} [t_{mm} + MR_{mm}(t_{vm})]$
 ↓
 miss rate

NOT: capacity (c): number
 of data bytes in cache
 block size (b): cache'e
 tek tek çekilen byte sayım
 num. of blocks ($B = c/b$)
 degree of ass. (N): number of
 blocks in a set
 number of sets ($S = B/N$)

NOT: L1 cache hızlı, küçük
 L2 " orta
 L3 " yavaş, büyük

NOT: Direct mapped cache
 $te \text{ MM}(\text{mod } N) = \text{cache}$
 ↓
 cache
 Sater sayısı