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
addi s0, s0, 0xEAB
```

```
#s0=0xFEDC9000
#s0=0xFEDC9000
  + 0xFFFFFEAB
  ------------
  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 2 'ler kaarp => 4 = s1
beq  s0,s1, target  # s0 = s1 ise target
addi s1,s1,1        } calismaz
sub  s1,s1,s0       }
```

```
target:
add s1,s1,s0        } s1 = s1 + s0 = 8
```

örnek ② branching

```
j target            # jump to target
srai s1,s1,2        }
addi s1,s1,1        } not executed
sub  s1,s1,s0       }
```

```
target:
add  s1,s1,s0       } s1 = s1 + s0
```

Loops brnek (1)

```
int sum=0;
int i;
for(i=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:
  slt  t2, s0, t0   #if s0<t0, t2=1 else t2=0
  beq  t2, zero, done
  add  s1, s1, s0
  slli  s0, s0, 1
  j  loop
done:
```

→ set less than

Arrays örnek ①

```
# s0=array base address , s1=i
lui s0,0x23B8F      #s0=0x23B8F000
ori s0,s0,0x400     #s0=0x23B8F400
addi s1,zero,0      #i=0
addi t2,zero,1000   #t2=1000

loop:
  bge s1,t2,done    # i<1000 old.süreç
  slli t0,s1,2      #t0=i*4
  add t0,t0,s0      # bir sonraki dizi elemanı (add. of array[i]
  lw t1,0(t0)       #t1=array[i]
  slli t1,t1,3      #t1=t1*8
  sw t1,0(t0)       #array[i]*=8
  addi s1,s1,1      # i++
  j loop

done:
```

Function calls örnek (1)

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

```
0x00000300 main: jal simple
0x00000304       add s0,s1,s2
```

call

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

```
0x0000051C simple: jr ra
```

return

jal simple:

ra = PC + 4 (0x00000304)

jumps to simple label (PC=0x0000051C)

jr ra:

PC = ra (0x00000304)

Function calls örnek (2)

```
int main(){
  int y;
  ...
  y= diffofsums(2,3,4,5); // 4 arguments
  ...
}
int diffofsums(int f, int g, int h, int i){
  int result;
  result =(f+g)-(h+i);
  return result;
}
```

NOT: diffofsums overwrote 3 reg. (t0, t1, s3)
can use stack to temporarily store registers.

```
#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
```

Örnek Direct mapped-cache

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

① 1024 adet blok var, 4KB toplam alan 4KB/1024 = 4 byte = 32 bit

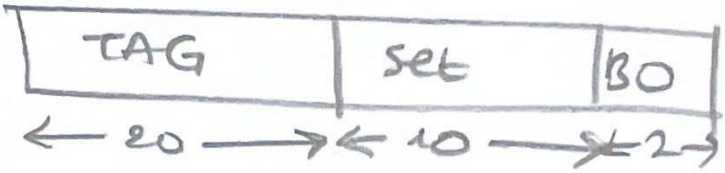
② Byte Offset = 1 word (blok genişliği) 4 byte, 1 word'ün belirli bir konuna erişebilmek için var.

$\log_2^4 = 2$ bit

③ set = hangi satırda olduğuna bakmak için var.

$\log_2^{1024} = 10$ 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

$\log_2^{16} = 4 \rightarrow 80$

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

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

TAG	set	80
50	10	4

③ 64 − 14 = TAG = 50

Ör: 2000 mem. access, 1250 cache'de var.
$t_{cache} = 1$ cycle, $t_{MM} = 100$ cycles.
cache hit and miss rate?
AMAT?

$\frac{1250}{2000} = 0,625$ hit rate $1 - 0,625 = 0,375$ miss rate

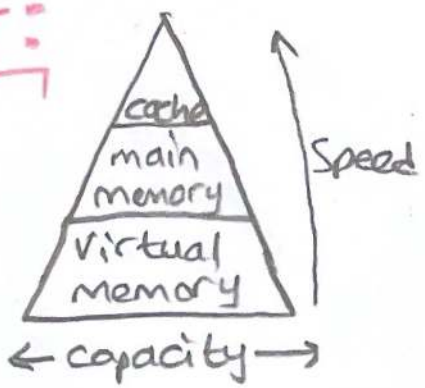
$1 + 0,375(100 + 0) = 38,5 = AMAT$

NOT: Multicycle'da
3 cycle beq
4 " R-type, addi, sw, jal
5 " lw

NOT: RF is faster than memory, writing memory is faster than reading memory

NOT: CPI = 4.12 cycle/ins.
T_{c_multi} = 375 ps
Exe. time = #ins * CPI * T_c

NOT:



NOT: Temporal locality:
Bir kez kullanılmışı tut
Spatial " :
" " kullanılmışın yanındakileri de 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'le
tekte çekilen byte sayısı
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 MM (mod N) = cache
↓
cache satır sayısı