Optimization

↳ Machine Independant opt.

↳ Code motion ⟶ loop içinde değeri hiç değişmeyen bir şeyi dışarı almak

↳ Reduction in strength ⟶ shift, add instead of multiply or divide

↳ Machine Dependant opt.

SUPERSCALAR PROCESSOR

Execute multiple instructions in one cycle

Superscalar Processor → executes multiple instructions in parallel by using multiple execution units

Pipeline execution → executes multiple instructions in the same execution unit in parallel by dividing the execution unit into different phases

Haswell CPU

Multiple instructions can execute in parallel

2 load with address computation

1 store with address computation

4 integer

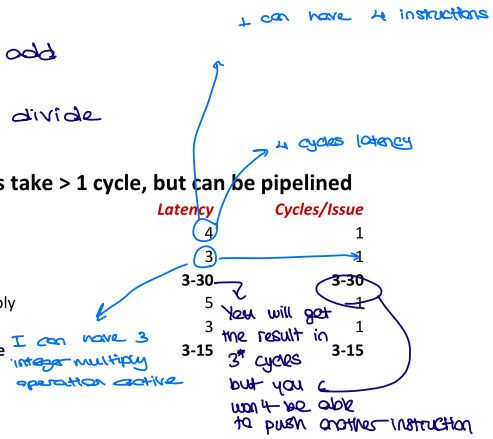
2 FP multiply

1 FP add

1 FP divide

- **Some instructions take > 1 cycle, but can be pipelined**

Instruction	*Latency*	*Cycles/Issue*
Load / Store	4	1
Integer Multiply	3	1
Integer/Long Divide	**3-30**	**3-30**
Single/Double FP Multiply	5	1
Single/Double FP Add	3	1
Single/Double FP Divide	**3-15**	**3-15**



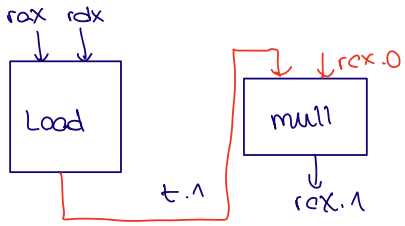
imulq (%rax, %rdx, 8), %rcx

load (%rax, %rdx .0, 4) → t.1 (temporary result)

imulq t.1, %rcx .0 → %rcx.1

incq %rdx → incq %rdx.0 %rdx.1

cmpq

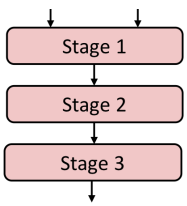


register %rdx changes on each iteration

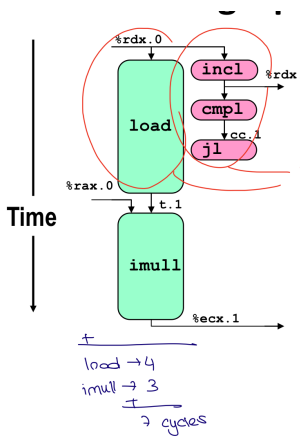
→ rename as %rdx.0 , %rdx.1 , %rdx.2 ...

Pipelined Functional Units

```
long mult_eg(long a, long b, long c) {
    long p1 = a*b;
    long p2 = a*c;
    long p3 = p1 * p2;
    return p3;
}
```



	Time						
	1	2	3	4	5	6	7
Stage 1	a*b	a*c			p1*p2		
Stage 2		a*b	a*c			p1*p2	
Stage 3			a*b	a*c			p1*p2




```
load    (%rax,%rdx,4) → t.1
imull   t.1, %ecx.0   → %ecx.1
addq $1,%rdx          → %rdx.1
cmpq %rdx.1, %rbp     → cc.1
jg-taken
```

dependency

dependency

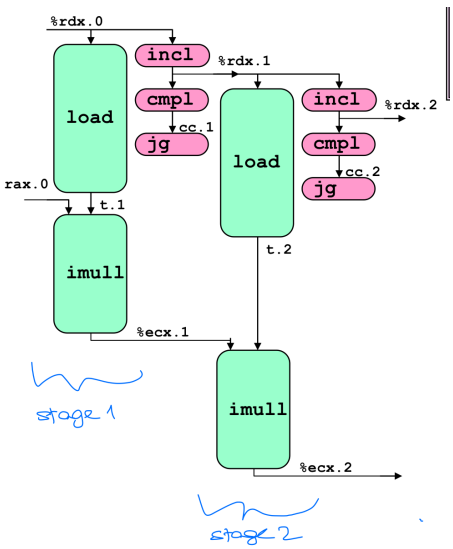
load ve diğer kısma aynı anda başlayabilirim

Operations

- Vertical position denotes time at which executed
  - Cannot begin operation until operands available
- Height denotes latency

Operands

- Arcs shown only for operands that are passed within execution unit



Selam ! :)

Tatatan

Nya