Mestrado em Informática

2007/08

A.J.Proença

Tema

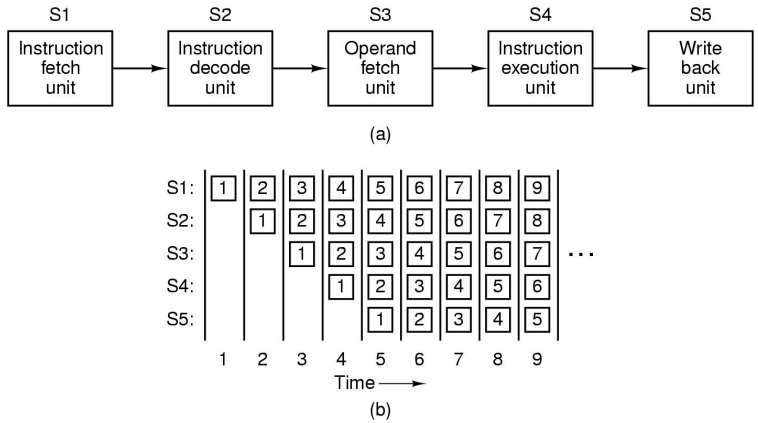
Revisitando os Sistemas de Computação (2)

Optimização do desempenho (no CPU)

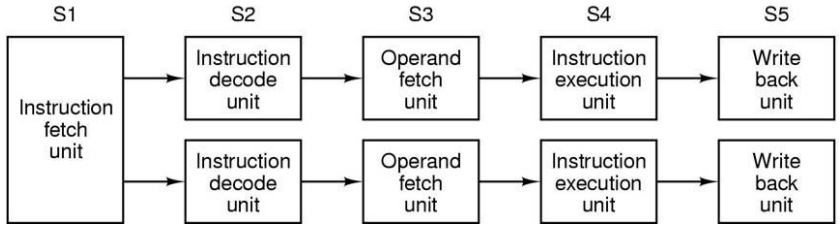
- com introdução de **paralelismo**
  - ao nível do processo (sistemas paralelos/distribuídos)
    - só fio de execução (multi-core...)
    - processo (memória partilhada/distribuída)
  - ao nível da instrução (*Instruction Level Parallelism*)
    - só nos dados (processadores vectoriais)
    - paralelismo desfasado (*pipeline*)
    - paralelismo "real" (superescalar)
  - no acesso à memória
    - paralelismo desfasado (*interleaving*)
    - paralelismo "real" (maior largura do *bus*)
    - paralelismo "redundante" (hierarquia de memória)

» **análise de processadores da Intel**

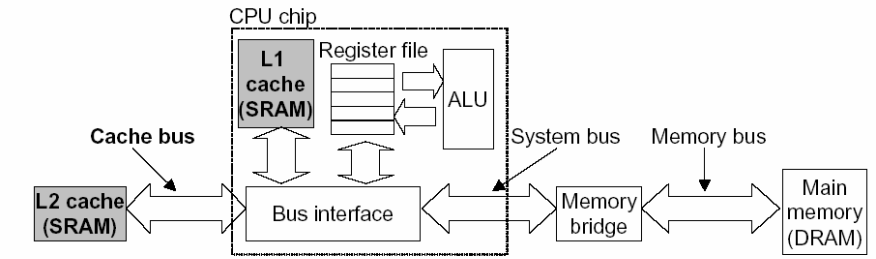
Exemplo de *pipeline*



Exemplo de superescalaridade (nível 2)



A introdução de cache na arquitectura Intel P6

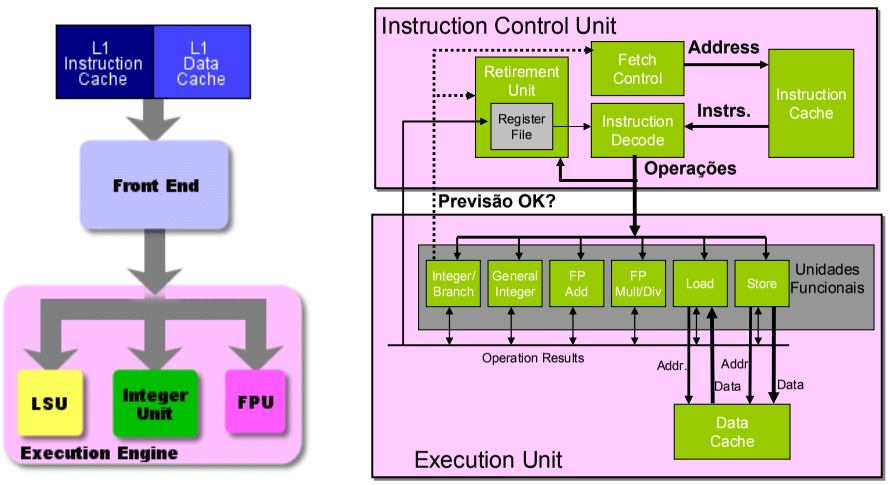


Nota: "Intel P6" é a designação comum da microarquitectura de PentiumPro, Pentium II e Pentium III

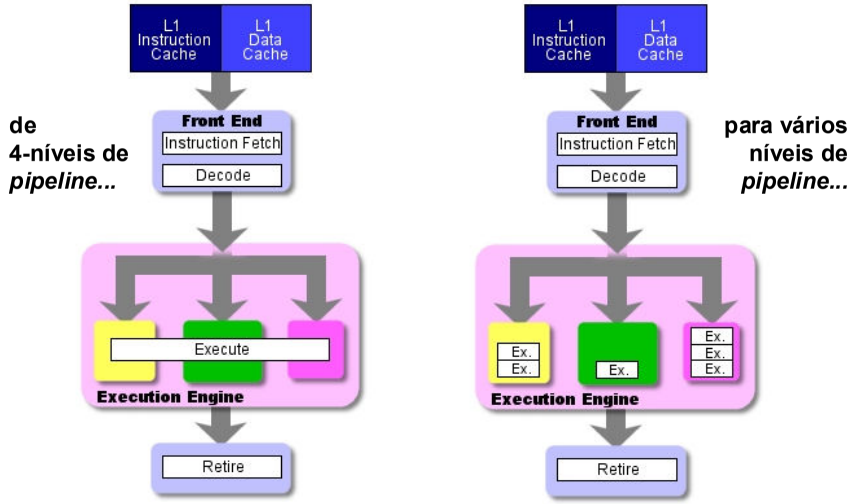
Para onde vão os processadores?

Intel marketing states that Core is a blend of P-M techniques and NetBurst architecture. However, Core is clearly a descendant of the Pentium Pro, or the P6 architecture. It is very hard to find anything "Pentium 4" or "NetBurst" in the Core architecture (...) it became clear that only the prefetching was inspired by experiences with the Pentium 4. Everything else is an evolution of "Yonah" (Core Duo), which was itself an improvement of Dothan and Banias. Those CPUs inherited the bus of the Pentium 4, but are still clearly children of the hugely successful P6 architecture. In a sense, you could call Core the "P8" architecture, with Banias/Dothan being based on the "P7" architecture.

A arquitectura interna dos processadores Intel P6 (1)



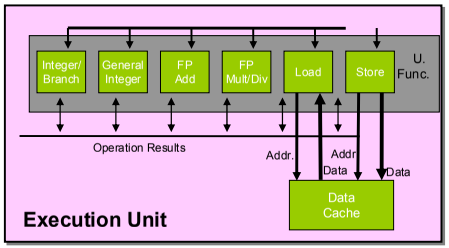
Algumas potencialidades do Intel P6 (1)



The main challenge for the CPU designer today is the average memory latency the CPU sees. A Pentium 4 3.6 GHz with DDR-400 runs (...) faster than the base clock of the RAM (200 MHz). (...) The result is that wait times of 200 to 300 cycles are not uncommon on the Pentium 4. The goal of CPU cache is to avoid accessing RAM, but (...)

Cache configuration	AMD K7 ("Barton")	AMD K8 ("Hammer")	Core	Intel NetBurst "Northwood"	Intel NetBurst "Prescott"	P-M Dothan	P6 ("Coppermine")
L1-cache (Instructions)	64 KB	64 KB	32 KB	8 KB	16 KB	32 KB	16 KB
L1-cache (Data)	64 KB	64 KB	32 KB	8 - 16 KB*	8-16 KB*	32 KB	16 KB
L1-cache latency (load to use)	3	3	3	2	4	3	3
L1 Associativity	2-way	2-way	8-way	4-way D, 8-way I	8-way	8-way	4-way
L1 I TLB (Entries)	24	32	128	128	128	128	32
L1 D TLB (Entries)	24	32	256	8	8	128	64
L2-cache (maximum)	512 KB	1 MB	4 MB	512 KB	2 MB	2 MB	256 KB
L2 Associativity	16-way	16-way	16-way	8-way	8-way	8-way	8-way
L2-cache Width	64-bit	128-bit	256-bit	256-bit	256-bit	256-bit	256-bit
L2-cache Latency load to use (+L1-latency)	16	12	11-20	11-20 (*)	27	12	7
L2 TLB (Entries)	256	512	n/a	64	64	n/a	n/a
Max. Memory Bandwidth to CPU	3.2 GB/s	6.4 GB/s	10.6 GB/s	4.2 GB/s	8.5 GB/s	4.2 GB/s	1.06 GB/s

- **Execução paralela de várias instruções**
  - 2 `integer` (1 pode ser `branch`)
  - 1 `FP Add`
  - 1 `FP Multiply ou Divide`
  - 1 `load`
  - 1 `store`

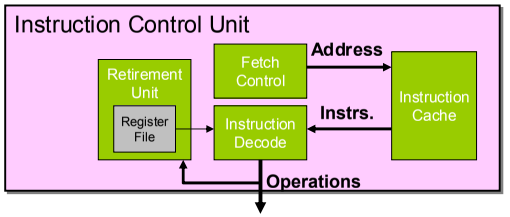


- Algumas instruções requerem > 1 ciclo, mas podem ser encadeadas

Instrução	Latência	Ciclos/Emissão
`Load / Store`	3	1
`Integer Multiply`	4	1
`Integer Divide`	36	36
`Double/Single FP Multiply`	5	2
`Double/Single FP Add`	3	1
`Double/Single FP Divide`	38	38

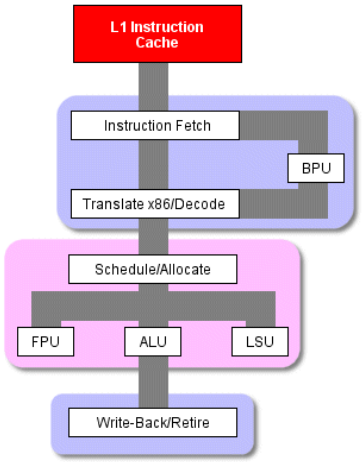
Papel da ICU:

- Lê instruções da *InstCache*
  - baseado no IP + previsão de saltos
  - antecipa dinamicamente (por *h/w*) se salta/não_salta e (possível) endereço de salto



- Traduz instruções em *Operações*
  - *Operações*: designação da Intel para instruções tipo-RISC
  - instrução típica requer 1–3 operações
- Converte referências a Registos em *Tags*
  - *Tags*: identificador abstracto que liga o resultado de uma operação com operandos-fonte de operações futuras

Algumas limitações do pipeline: dependências de dados e saltos condicionais




```
incl   %edx
cmpl   %esi,%edx
jl     .L24
```

Ver exemplo nos slides seguintes...

Análise detalhada do exemplo: forma genérica e abstracta de `combine`

```
void abstract_combine4(vec_ptr v, data_t *dest)
{
  int i;
  int length = vec_length(v);
  data_t *data = get_vec_start(v);
  data_t t = IDENT;
  for (i = 0; i < length; i++)
    t = t OP data[i];
  *dest = t;
}
```

- **Procedimento**
  - calcula a soma de todos os elementos do vector
  - guarda o resultado numa localização destino
  - estrutura e operações do vector definidos via ADT

Conversão de instruções com registos para operações com tags

- **Versão de `combine4`**
  - tipo de dados: *inteiro* ; operação: *multiplicação*

```
.L24:                          # Loop:
  imull  (%eax,%edx,4),%ecx    # t *= data[i]
  incl   %edx                  # i++
  cmpl   %esi,%edx             # i:length
  jl     .L24                  # if < goto Loop
```

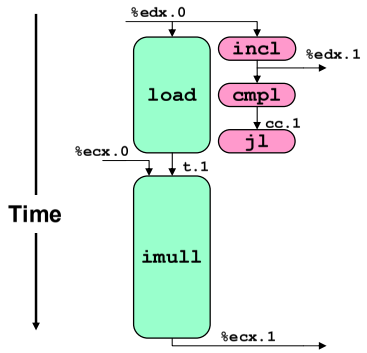
- **Tradução da 1ª iteração**

```
.L24:
  imull  (%eax,%edx,4),%ecx

  incl   %edx
  cmpl   %esi,%edx
  jl     .L24
```

```
load   (%eax,%edx.0,4) ➔ t.1
imull  t.1, %ecx.0     ➔ %ecx.1
incl   %edx.0          ➔ %edx.1
cmpl   %esi, %edx.1    ➔ cc.1
jl     -taken cc.1
```

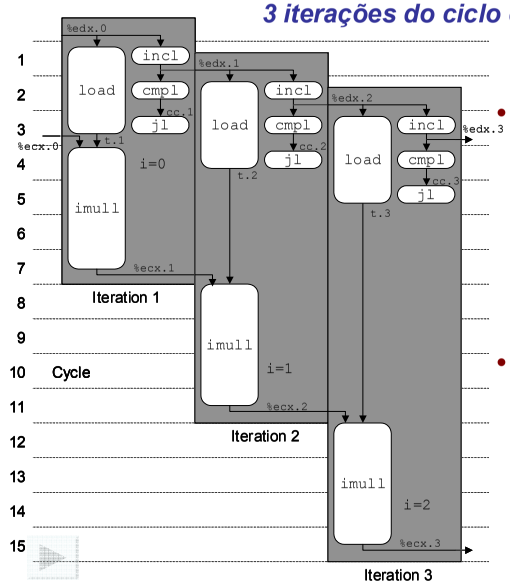

Análise visual da execução de instruções no P6: 1 iteração do ciclo de produtos em `combine`




```
load   (%eax,%edx.0,4) ➔ t.1
imull  t.1, %ecx.0     ➔ %ecx.1
incl   %edx.0          ➔ %edx.1
cmpl   %esi, %edx.1    ➔ cc.1
jl     -taken cc.1
```

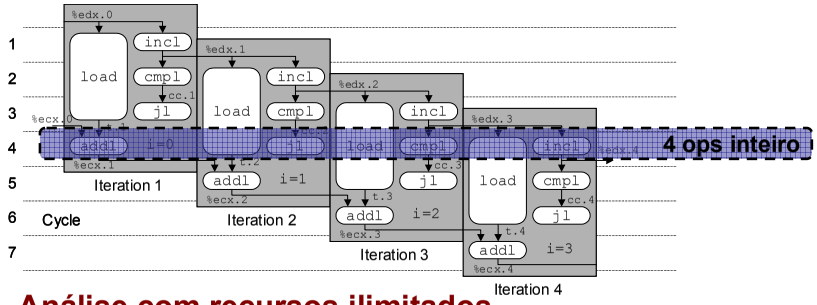
- **Operações**
  - a posição vertical dá uma indicação do tempo em que é executada
    - uma operação não pode iniciar-se sem os seus operandos
  - a altura traduz a latência
- **Operandos**
  - os arcos apenas são representados para os operandos que são usados no contexto da *execution unit*

Análise visual da execução de instruções no P6: 3 iterações do ciclo de produtos em `combine`



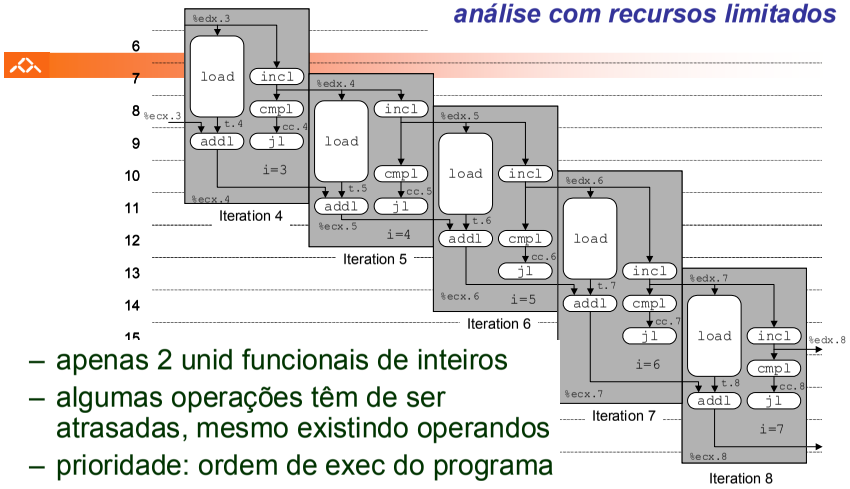
- **Análise com recursos ilimitados**
  - execução paralela e encadeada de operações na EU
  - execução *out-of-order* e especulativa
- **Desempenho**
  - factor limitativo: latência da multipl. de inteiros
  - CPE: 4.0

Análise visual da execução de instruções no P6: 4 iterações do ciclo de somas em `combine`



- **Análise com recursos ilimitados**
- **Desempenho**
  - pode começar uma nova iteração em cada ciclo de *clock*
  - valor teórico de CPE: 1.0
  - requer a execução de 4 operações c/ inteiros em paralelo

As iterações do ciclo de somas: análise com recursos limitados



- apenas 2 unid funcionais de inteiros
- algumas operações têm de ser atrasadas, mesmo existindo operandos
- prioridade: ordem de exec do programa

- **Desempenho**
  - CPE expectável: 2.0

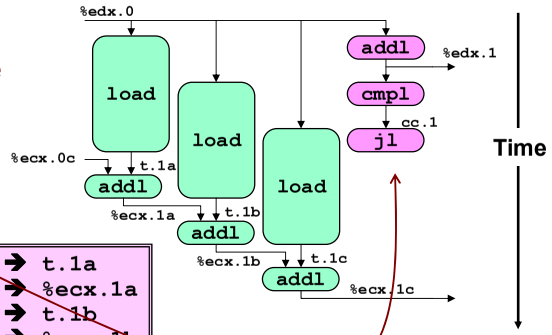
Técnicas de optimização dependentes da máquina: loop unroll (1)

```
void combine5(vec_ptr v, int *dest)
{
  int length = vec_length(v);
  int limit = length-2;
  int *data = get_vec_start(v);
  int sum = 0;
  int i;
  /* junta 3 elem's no mesmo ciclo */
  for (i = 0; i < limit; i+=3) {
    sum += data[i] + data[i+1]
           + data[i+2];
  }
  /* completa os restantes elem's */
  for (; i < length; i++) {
    sum += data[i];
  }
  *dest = sum;
}
```

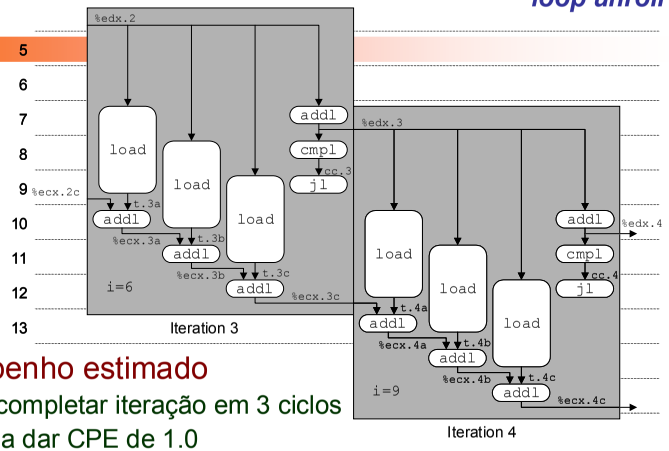
Optimização 4:

- juntar várias (**3**) iterações num simples ciclo
- amortiza *overhead* dos ciclos em várias iterações
- termina extras no fim
- **CPE: 1.33**

- **loads** podem encadear, uma vez que não há dependências
- apenas um conjunto de instruções de controlo de ciclo




```
load (%eax,%edx.0,4)    ➔ t.1a
iaddl t.1a, %ecx.0c     ➔ %ecx.1a
load 4(%eax,%edx.0,4)   ➔ t.1b
iaddl t.1b, %ecx.1a     ➔ %ecx.1b
load 8(%eax,%edx.0,4)   ➔ t.1c
iaddl t.1c, %ecx.1b     ➔ %ecx.1c
iaddl $3,%edx.0         ➔ %edx.1
cmpl %esi, %edx.1       ➔ cc.1
jl-taken cc.1
```

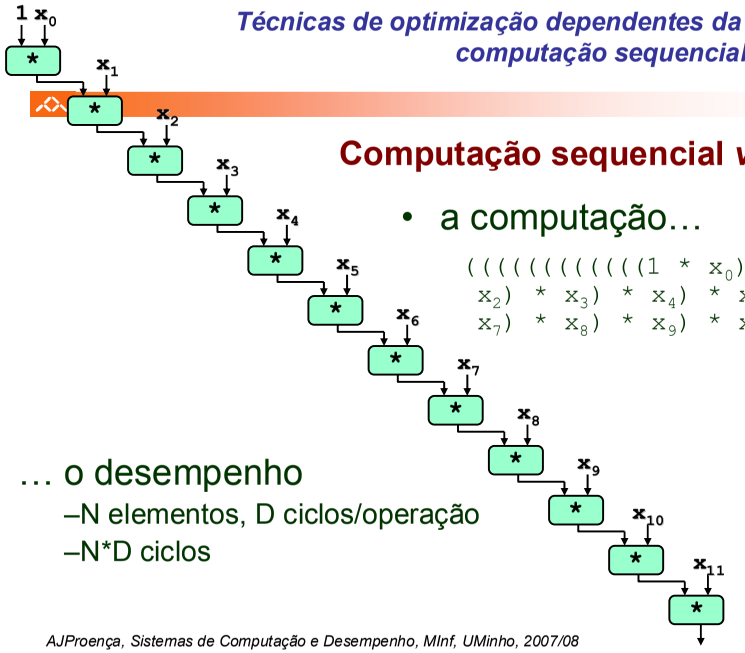



- Desempenho estimado
  - pode completar iteração em 3 ciclos
  - deveria dar CPE de 1.0
- Desempenho medido
  - CPE: 1.33
  - 1 iteração em cada 4 ciclos

Valor do **CPE** para várias situações de *loop unroll*:

Grau de *Unroll*		1	2	3	4	8	16
Inteiro	Soma	2.00	1.50	1.33	1.50	1.25	1.06
Inteiro	Produto	4.00					
fp	Soma	3.00					
fp	Produto	5.00					

- apenas melhora nas somas de inteiros
  - restantes casos há restrições com a latência da unidade
- efeito não é linear com o grau de *unroll*
  - há efeitos subtis que determinam a atribuição exacta das operações



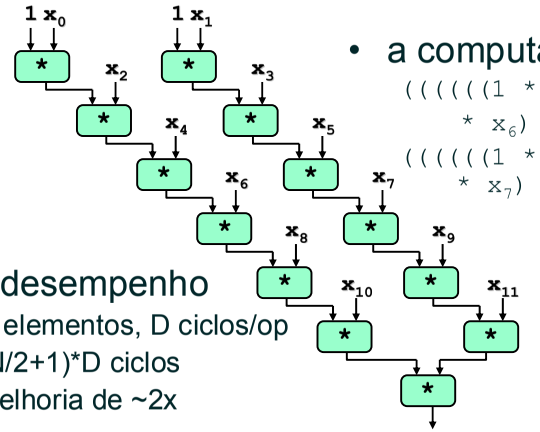
Computação sequencial *versus* ...

- a computação…

$$((((((((((((1 * x_0) * x_1) * x_2) * x_3) * x_4) * x_5) * x_6) * x_7) * x_8) * x_9) * x_{10}) * x_{11})$$

… o desempenho
- N elementos, D ciclos/operação
- N*D ciclos

Computação sequencial ... *versus* paralela!



- a computação…

$$((((((1 * x_0) * x_2) * x_4) * x_6) * x_8) * x_{10}) *$$
$$((((((1 * x_1) * x_3) * x_5) * x_7) * x_9) * x_{11})$$

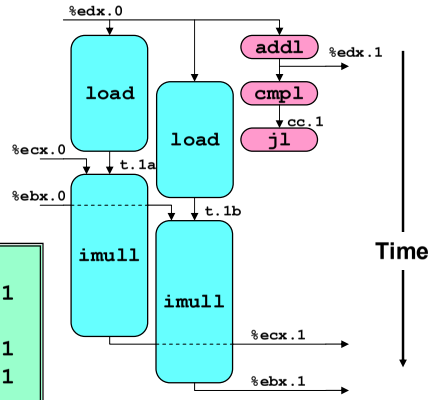
… o desempenho
- –N elementos, D ciclos/op
- –(N/2+1)*D ciclos
- –melhoria de ~2x


```
void combine6(vec_ptr v, int *dest)
{
  int length = vec_length(v);
  int limit = length-1;
  int *data = get_vec_start(v);
  int x0 = 1;
  int x1 = 1;
  int i;
  /* junta 2 elem's de cada vez */
  for (i = 0; i < limit; i+=2) {
    x0 *= data[i];
    x1 *= data[i+1];
  }
  /* completa os restantes elem's */
  for (; i < length; i++) {
    x0 *= data[i];
  }
  *dest = x0 * x1;
}
```

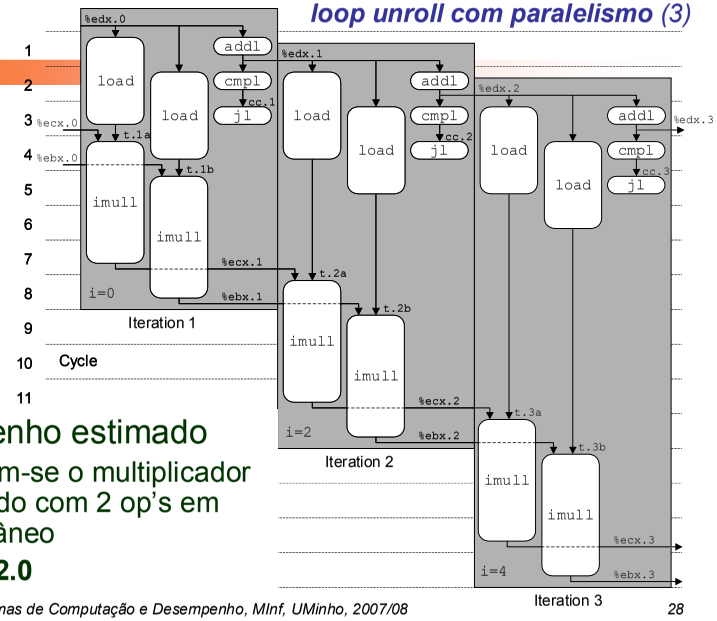
... *versus* paralela!

Optimização 5:
- –acumular em 2 produtos diferentes
  - pode ser feito em paralelo, se OP fôr associativa!
- –juntar no fim
- –Desempenho
  - –**CPE: 2.0**
  - –melhoria de 2x

- os dois produtos no interior do ciclo não dependem um do outro…
- e é possível encadeá-los
- *iteration splitting*, na literatura




```
load (%eax,%edx.0,4)   → t.1a
imull t.1a, %ecx.0     → %ecx.1
load 4(%eax,%edx.0,4)  → t.1b
imull t.1b, %ebx.0     → %ebx.1
iaddl $2,%edx.0        → %edx.1
cmpl %esi, %edx.1      → cc.1
jl-taken cc.1
```

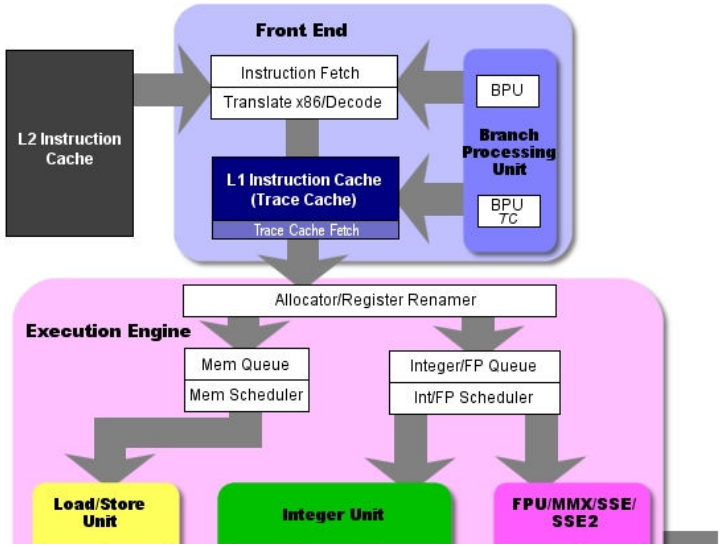
Desempenho estimado
- – mantém-se o multiplicador ocupado com 2 op's em simultâneo
- – **CPE: 2.0**

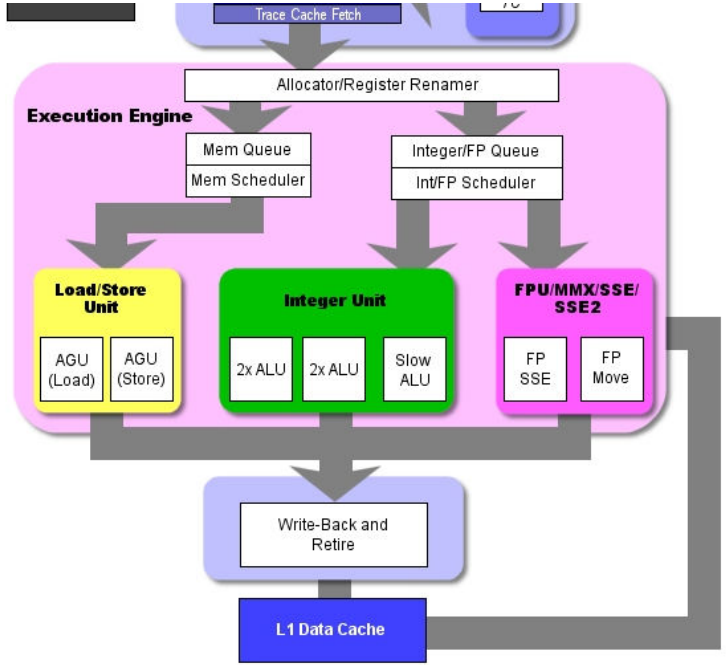
Método	Inteiro		Real (precisão simples)	
	+	*	+	*
Abstract -g	42.06	41.86	41.44	160.00
Abstract -O2	31.25	33.25	31.25	143.00
Move `vec_length`	20.66	21.25	21.15	135.00
Acesso aos dados	6.00	9.00	8.00	117.00
Acum. em temp	2.00	4.00	3.00	5.00
Unroll 4x	1.50	4.00	3.00	5.00
Unroll 16x	1.06	4.00	3.00	5.00
Unroll 2x, paral. 2x	1.50	2.00	2.00	2.50
Unroll 4x, paral. 4x	1.50	2.00	1.50	2.50
Unroll 8x, paral. 4x	1.25	1.25	1.50	2.00
Optimização Teórica	1.00	1.00	1.00	2.00
Pior : Melhor	*39.7*	*33.5*	*27.6*	*80.0*

- Precisa de muitos registos!
  - para guardar somas/produtos
  - apenas 6 registos (p/ inteiros) disponíveis no IA32
    - tb usados como apontadores, controlo de ciclos, …
  - 8 registos de fp
  - quando os registos são insuficientes, temp's vão para a *stack*
    - elimina ganhos de desempenho
      (ver *assembly* em produto inteiro com *unroll* 8x e paralelismo 8x)
  - re-nomeação de registos não chega
    - não é possível referenciar mais operandos que aqueles que o *instruction set* permite
    - … principal inconveniente do *instruction set* do IA32
- Operações a paralelizar têm de ser associativas
  - a soma e multipl de fp num computador não é associativa!
    - (3.14+1e20)-1e20 nem sempre é igual a 3.14+(1e20-1e20)…

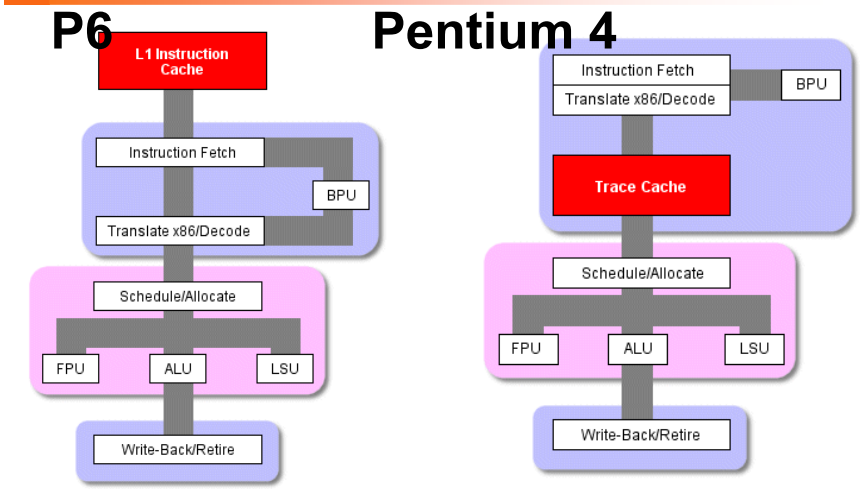
- **`combine`**
  - produto de inteiros
  - *unroll* 8x e paralelismo 8x
  - 7 variáveis locais partilham 1 registo (`%edi`)
    - observar os acessos à *stack*
    - melhoria desempenho é comprometida...
    - *register spilling* na literatura

```
.L165:
      imull (%eax),%ecx
      movl -4(%ebp),%edi
      imull 4(%eax),%edi
      movl %edi,-4(%ebp)
      movl -8(%ebp),%edi
      imull 8(%eax),%edi
      movl %edi,-8(%ebp)
      movl -12(%ebp),%edi
      imull 12(%eax),%edi
      movl %edi,-12(%ebp)
      movl -16(%ebp),%edi
      imull 16(%eax),%edi
      movl %edi,-16(%ebp)
   ...
      addl $32,%eax
      addl $8,%edx
      cmpl -32(%ebp),%edx
      jl .L165
```

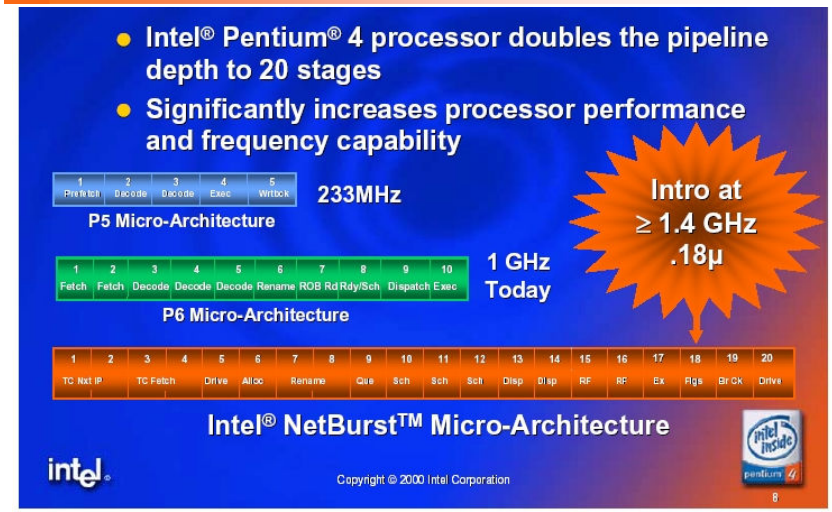





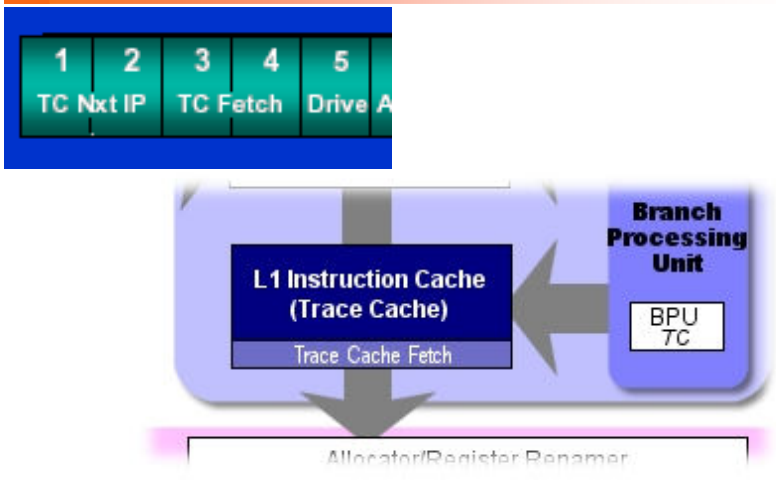
Percurso crítico no pipeline de instruções: o P6 e o Pentium 4



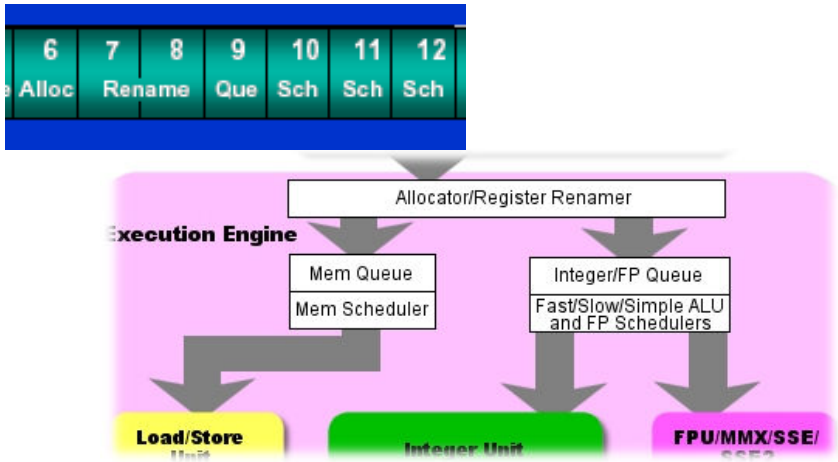
Níveis de pipeline em 3 gerações de Pentium



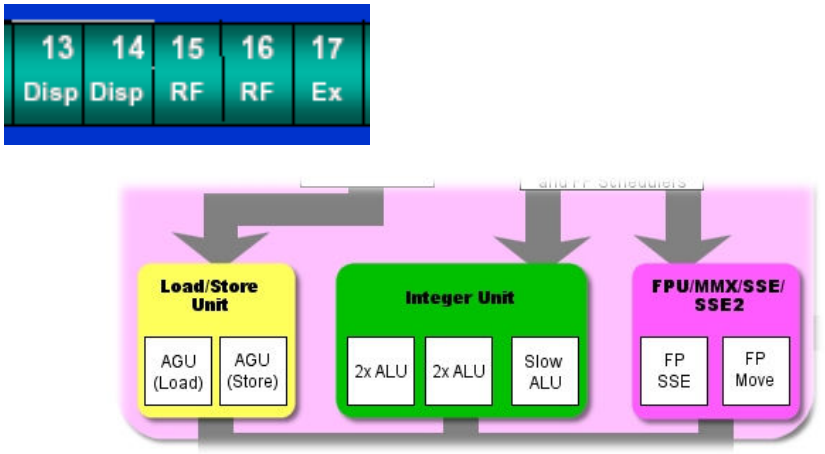
O pipeline no Pentium 4: níveis 1-5



O pipeline no Pentium 4: níveis 6-12



O pipeline no Pentium 4: níveis 13-17



O pipeline no Pentium 4: níveis 18-20