Organização de Computadores I
DCC006

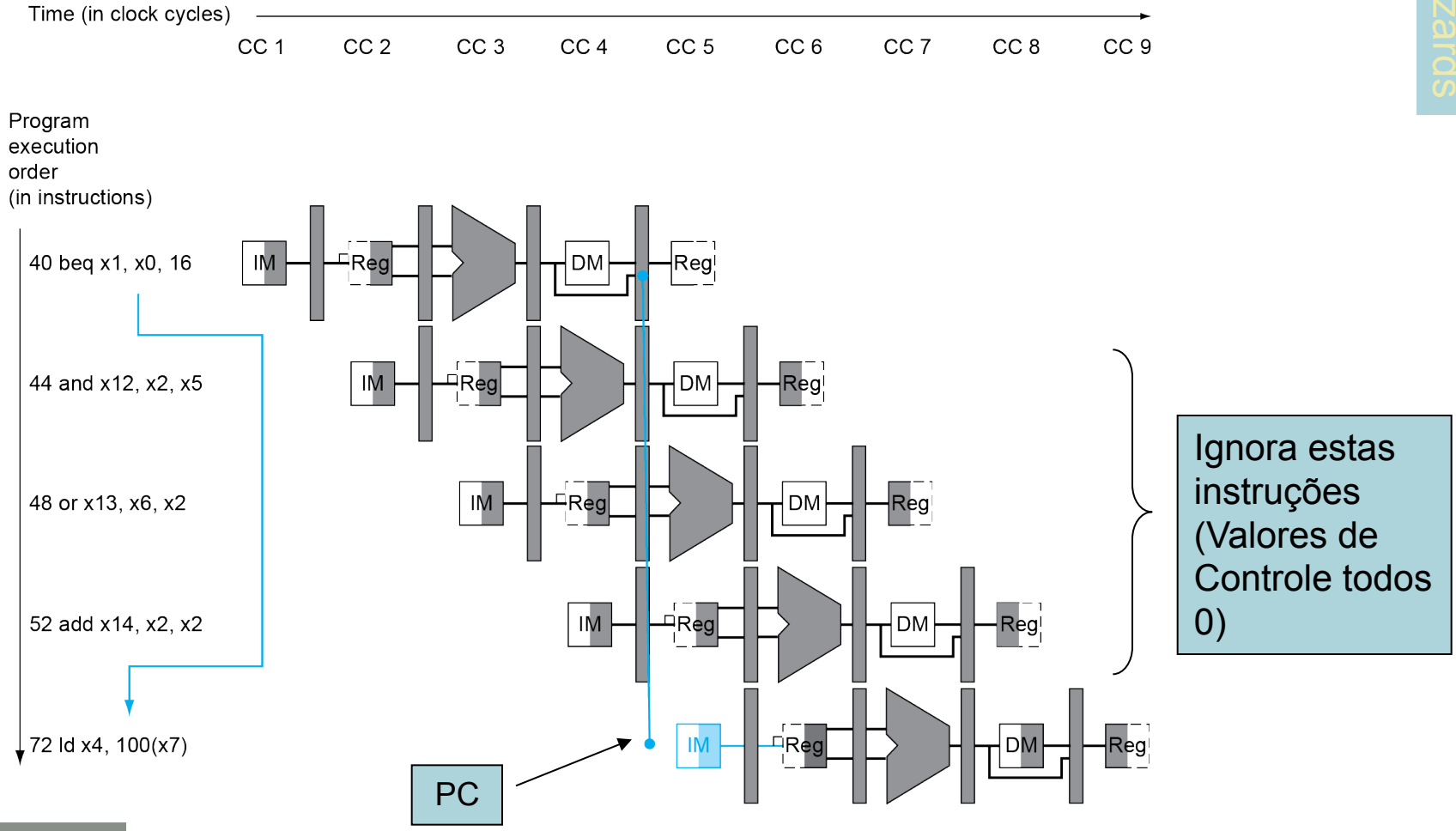
Aula 10 – O Processador – Pipeline

Prof. Omar Paranaiba Vilela Neto

UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Hazards de Desvio

- Se a saída do branch é determinada em MEM

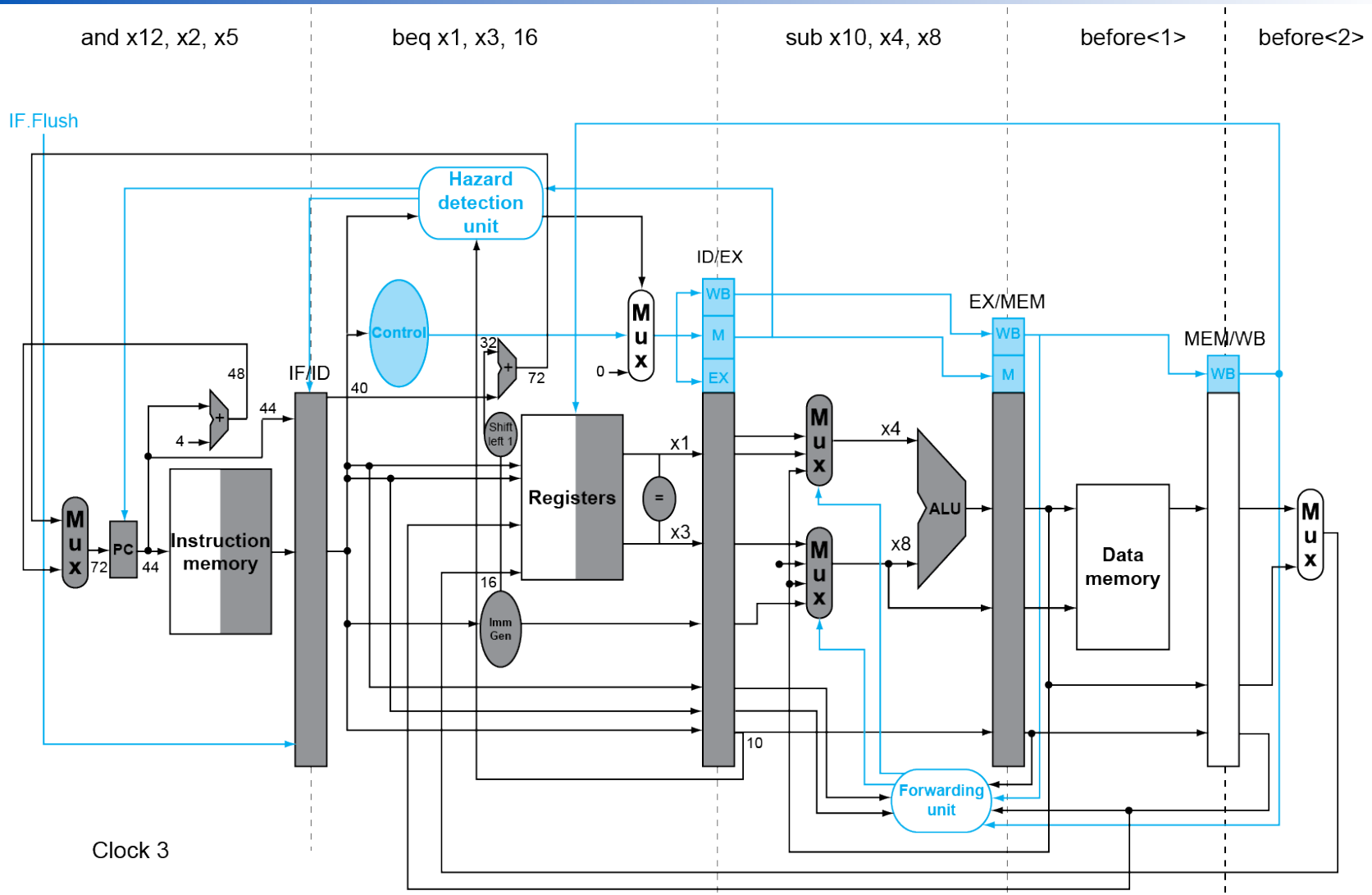


Reduzindo atraso do Branch

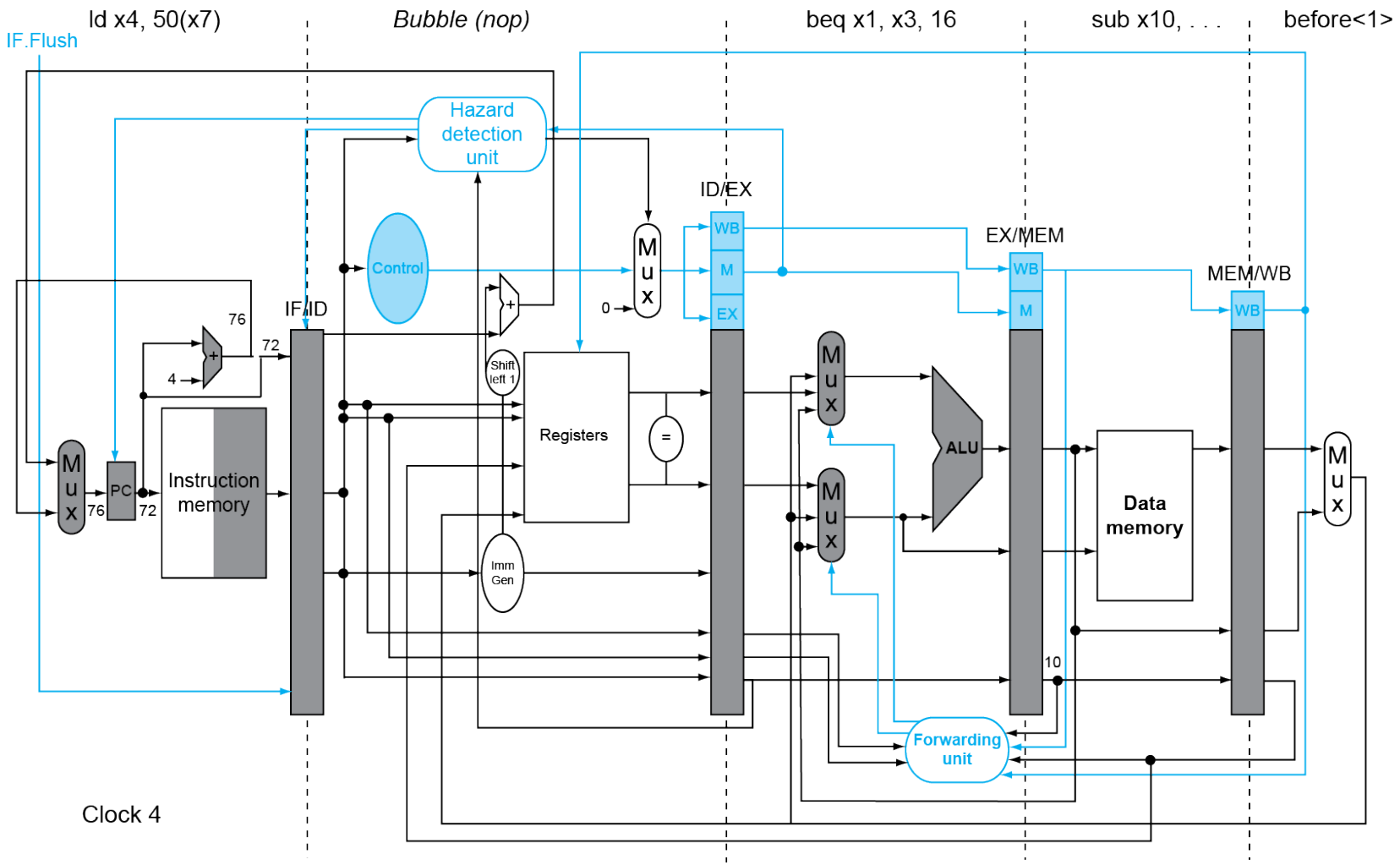
- Move o hardware de detecção para o estágio ID
  - Somador de endereço de destino
  - Registrador de comparação
- Exemplo: branch tomado

```
36:   sub   x10, x4, x8
40:   beq   x1,  x3, 16   // PC-relative branch
            stall         // to 40+16*2=72
44:   and   x12, x2, x5
48:   orr   x13, x2, x6
52:   add   x14, x4, x2
56:   sub   x15, x6, x7
      ...
72:   ld    x4, 50(x7)
```

Exemplo: Branch tomado



Exemplo: Branch tomado

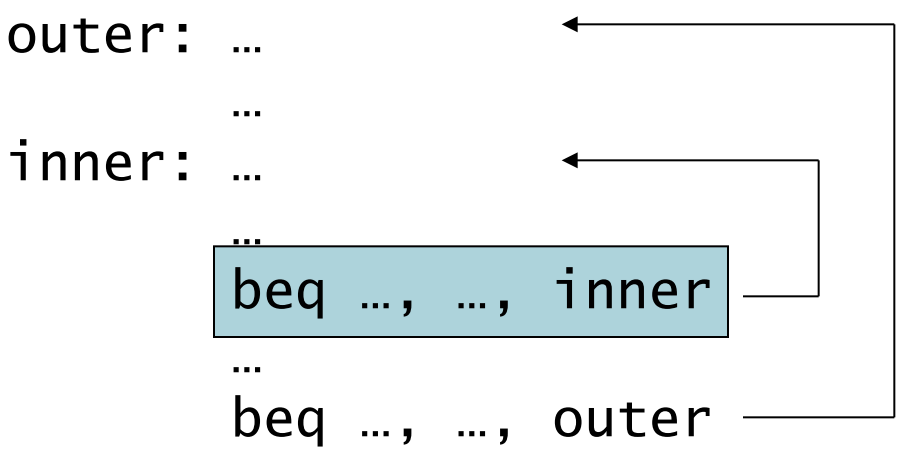


Previsão dinâmica de Branch

- Em pipelines profundos e superescalar, penalidade do branch é mais significativa
- Usa previsão dinâmica
  - Buffer de previsão (branch history table)
  - Indexado pelo endereço da instrução branch
  - Guarda saída anterior (tomado/não tomado)
  - Para executar um branch
    - Verifique a tabela, espere a mesma saída anterior
    - Comece a busca considerando a previsão
    - Se errado, corrije o pipeline e muda previsão

1-Bit Previsor: Saída

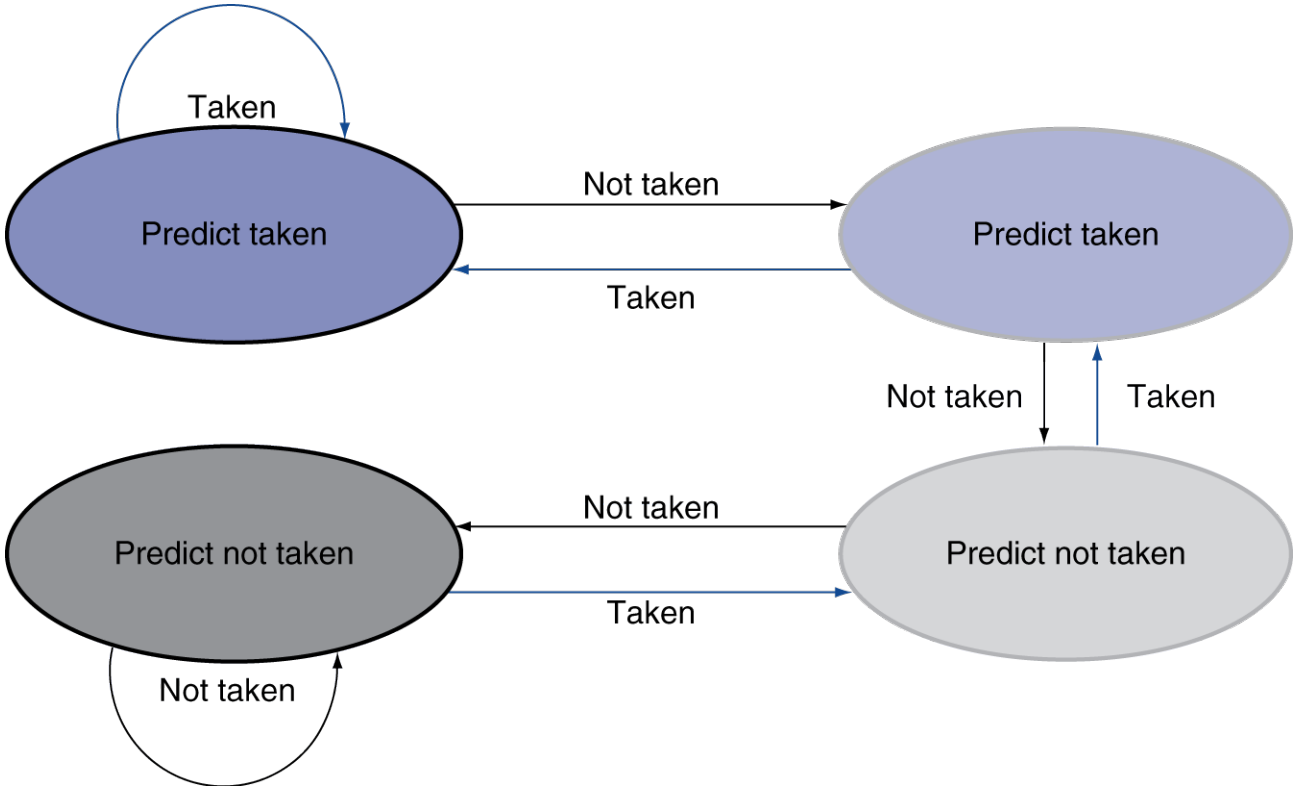
- Loop interno prevê errado duas vezes!



  - Erra na última interação do loop
  - Erra na primeira interação da próxima vez no loop

2-Bit Previsor

- Somente troca previsor quando errar duas vezes consecutivas



Calculando o destino do Branch

- Mesmo prevendo correto, precisa calcular o endereço de destino
  - 1-ciclo de penalidade para previsão tomado
- Branch target buffer
  - Guarda o endereço de destino
  - Indexado pelo PC quando instrução buscada
    - Busca endereço de destino imediatamente

Instruction-Level Parallelism (ILP)

- Pipelining: executa múltiplas instruções em paralelo
- Para aumentar ILP
  - Pipeline mais profundo
    - Menos trabalho por estágio ⇒ ciclo de clock menos
  - Múltipla tarefas
    - Replica estágios do pipeline ⇒ múltiplos pipelines
    - Começa múltiplas instruções por clock
    - CPI < 1, usar Instruções por Ciclo (IPC)
    - Ex., 4GHz 4-way multiple-issue
      - 16 BIPS, pico CPI = 0.25, pico IPC = 4
    - Porém, dependência reduzem o ganho na prática

Múltiplas Tarefas

- Múltiplas tarefas - Estática
  - Compilador diz quais instruções disparar juntas
  - Empacota em “slots de instruções”
  - Compilador detecta e evita hazards
- Múltiplas tarefas - Dinâmica
  - CPU examina “grupo” de instruções e escolhe as que podem disparar juntas
  - Compilador pode ajudar reordenando instruções
  - CPU resolve hazards usando técnicas avançadas em tempo de execução

Especulação

- “Chuta” o que fazer com uma instrução
  - Começa operação o mais rápido possível
  - Verifica se o chute está certo
    - Se certo, completa operação
    - Se errado, retorna e faz o certo
- Comum em múltiplas tarefas estática e dinâmica
- Exemplos
  - Especula na saída do Branch (Previsão)
    - Retorne se estiver errado
  - Especula no load
    - Retorna se localização for atualizada

Especulação Compilador/Hardware

- Compilador pode reordenar instruções
  - ex., move load para antes do branch
  - Pode incluir instruções de correção para recuperar chutes errados
- Hardware pode olhar adiante para instruções que executam
  - Guarda resultados até determinar se são necessárias
  - Ignoram em caso de chute errado

Múltiplas Tarefas - Estática

- Compilador agrupa instruções em "pacotes de disparo"
  - Grupo de instruções que podem ser disparadas em um único ciclo
  - Determinado pelos requisitos necessários pelo pipeline
- Pense em um pacote de disparo como uma instrução longa
  - Especifica múltiplas instruções concorrentes
  - ⇒ Very Long Instruction Word (VLIW)

Agendando Múltiplas Tarefas - Estática

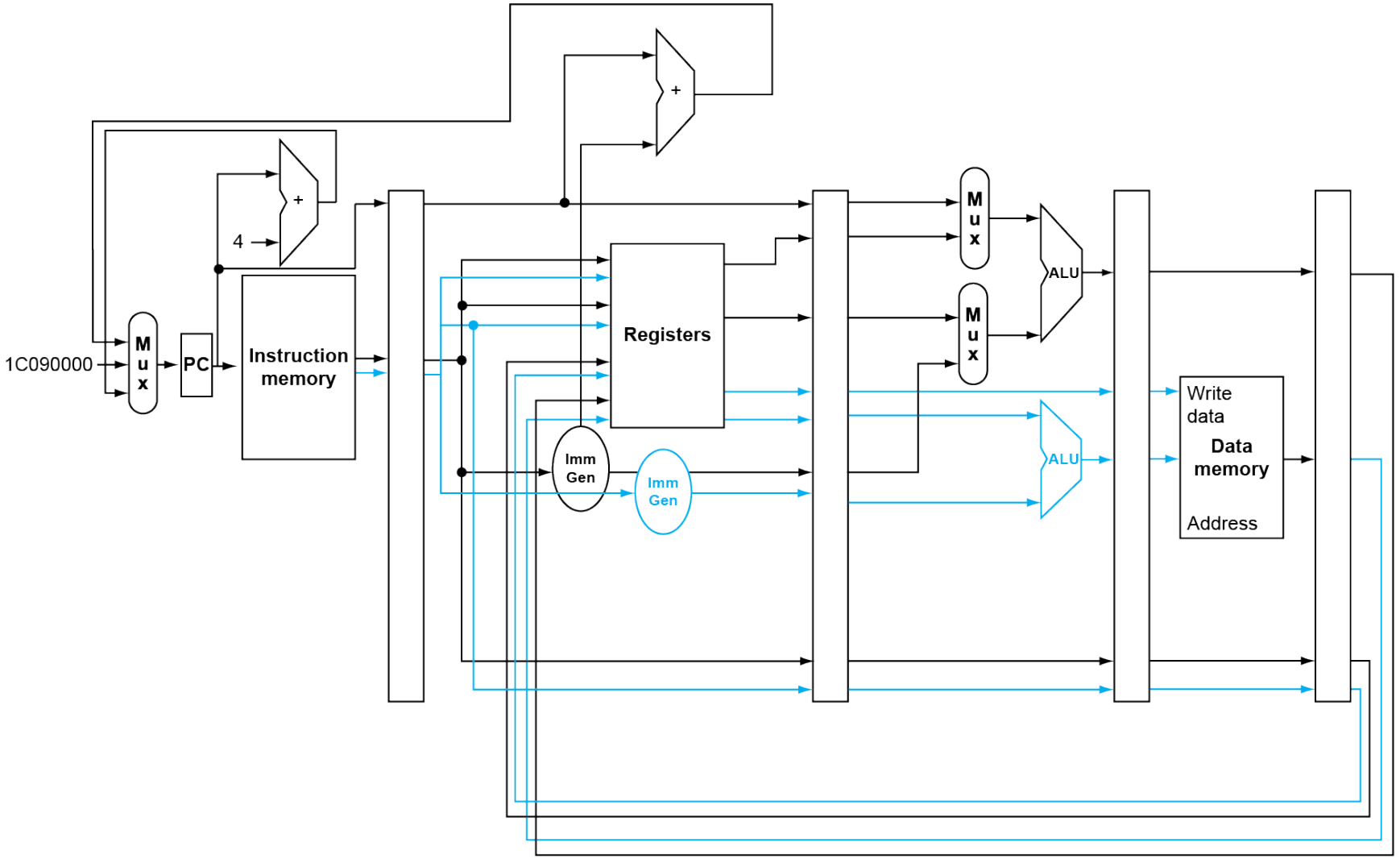
- Compilador deve resolver hazards
  - Reordena instruções em pacotes de disparo
  - Sem dependências no pacote
  - Usa NOPse necessário

RISC-V com tarefas duplas

- Pacotes de duas tarefas
  - Uma instrução ALU ou branch
  - Uma instrução load ou store
  - 64-bits alinhados
    - ALU/branch, e então load/store
    - NOP quando necessário

Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

RISC-V com tarefas duplas



RISC-V com tarefas duplas - Hazards

- Mais instruções executando em paralelo
- EX Hazard de dados
  - Impossibilita encaminhamento no mesmo pacote
  - Não pode usar resultado da ALU no load/store
    - ```
      add  x10, x0, x1
      ld   x2, 0(x10)
      ```
    - Separe em dois pacotes diferentes
- Load-use hazard
  - Continua um ciclo de stall, mas agora duas instruções
- Agendamento mais agressivo é necessário
      ```

Exemplo de agendamento

- RISC-V de duas tarefas

```
Loop: ld   x31,0(x20)     // x31=array element
      add  x31,x31,x21    // add scalar in x21
      sd   x31,0(x20)     // store result
      addi x20,x20,-8     // decrement pointer
      blt  x22,x20,Loop   // branch if x22 < x20
```

	ALU/branch	Load/store	cycle
Loop:	nop	ld x31,0(x20)	1
	addi x20,x20,-8	nop	2
	add x31,x31,x21	nop	3
	blt x22,x20,Loop	sd x31,8(x20)	4

- IPC = 5/4 = 1.25 (pico IPC = 2)

Múltiplas Tarefas - Dinâmica

- Processador “Superscalar”
- CPU decide quando disparar 0, 1, 2, ... Em cada ciclo
  - Evitando hazards estrutural e de dados
- Evita que o compilador agende
  - Mas ele ainda pode ajudar
  - Semântica de código garantido pela CPU

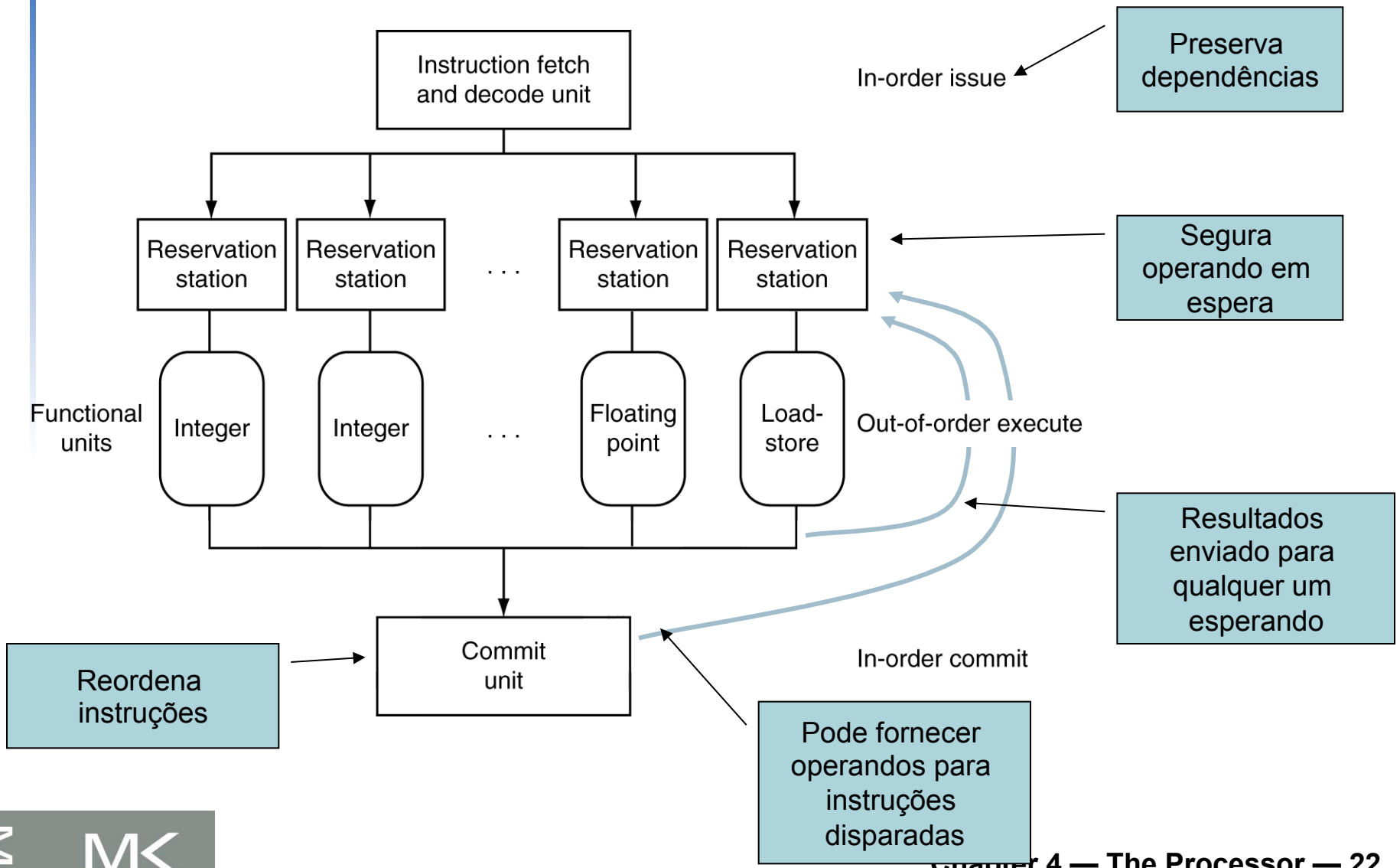
Agendamento dinâmico do pipeline

- Permite que CPU execute instruções fora de ordem para evitar stalls
- Exemplo

```
ld   x31,20(x21)
add  x1,x31,x2
sub  x23,x23,x3
andi x5,x23,20
```

  - Pode começar o sub enquanto add está esperando o ld

Agendamento dinâmico do pipeline



Por que agendar dinamicamente?

- Por que não deixar o compulador agendar?
- Nem tudo pode ser previsto antes
  - ex., cache misses
- Implementações diferentes de um ISA tem latências e hazards diferentes

Múltiplas tarefas funciona?

The BIG Picture

- Sim, mas não muito como gostaríamos
- Dependências reais limitam paralelismo
- Algumas dependências são difíceis de eliminar
- Paralelismo nem sempre é exposto
- Atraso de memória são limitados por largura de banda
- Especulação ajuda quando bem feito

Eficiência de Energia

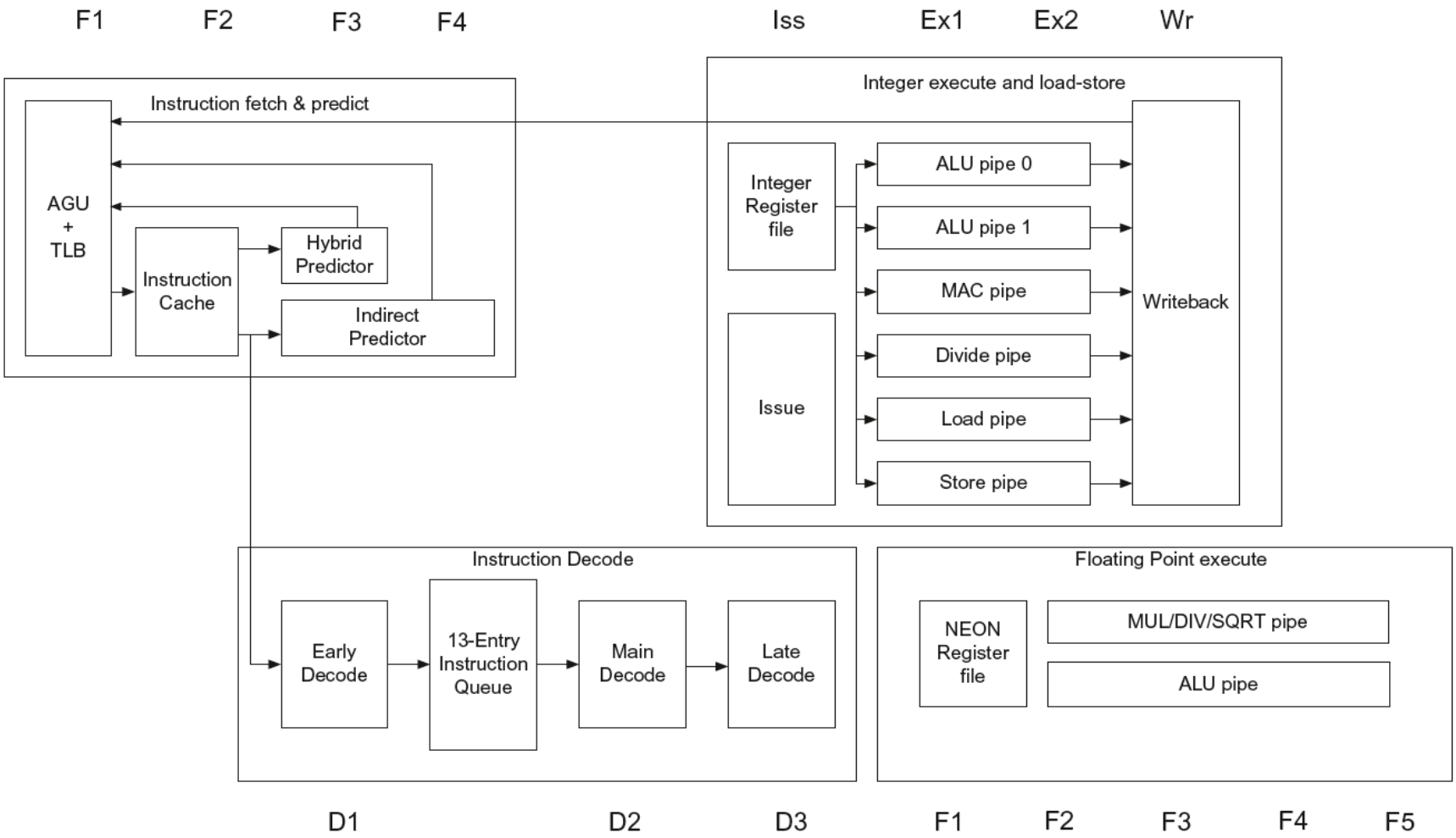
- Complexidade do agendamento e especulação requer energia
- Múltiplos cores simples podem ser melhor

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue width	Out-of-order/ Speculation	Cores	Power
i486	1989	25MHz	5	1	No	1	5W
Pentium	1993	66MHz	5	2	No	1	10W
Pentium Pro	1997	200MHz	10	3	Yes	1	29W
P4 Willamette	2001	2000MHz	22	3	Yes	1	75W
P4 Prescott	2004	3600MHz	31	3	Yes	1	103W
Core	2006	2930MHz	14	4	Yes	2	75W
UltraSparc III	2003	1950MHz	14	4	No	1	90W
UltraSparc T1	2005	1200MHz	6	1	No	8	70W

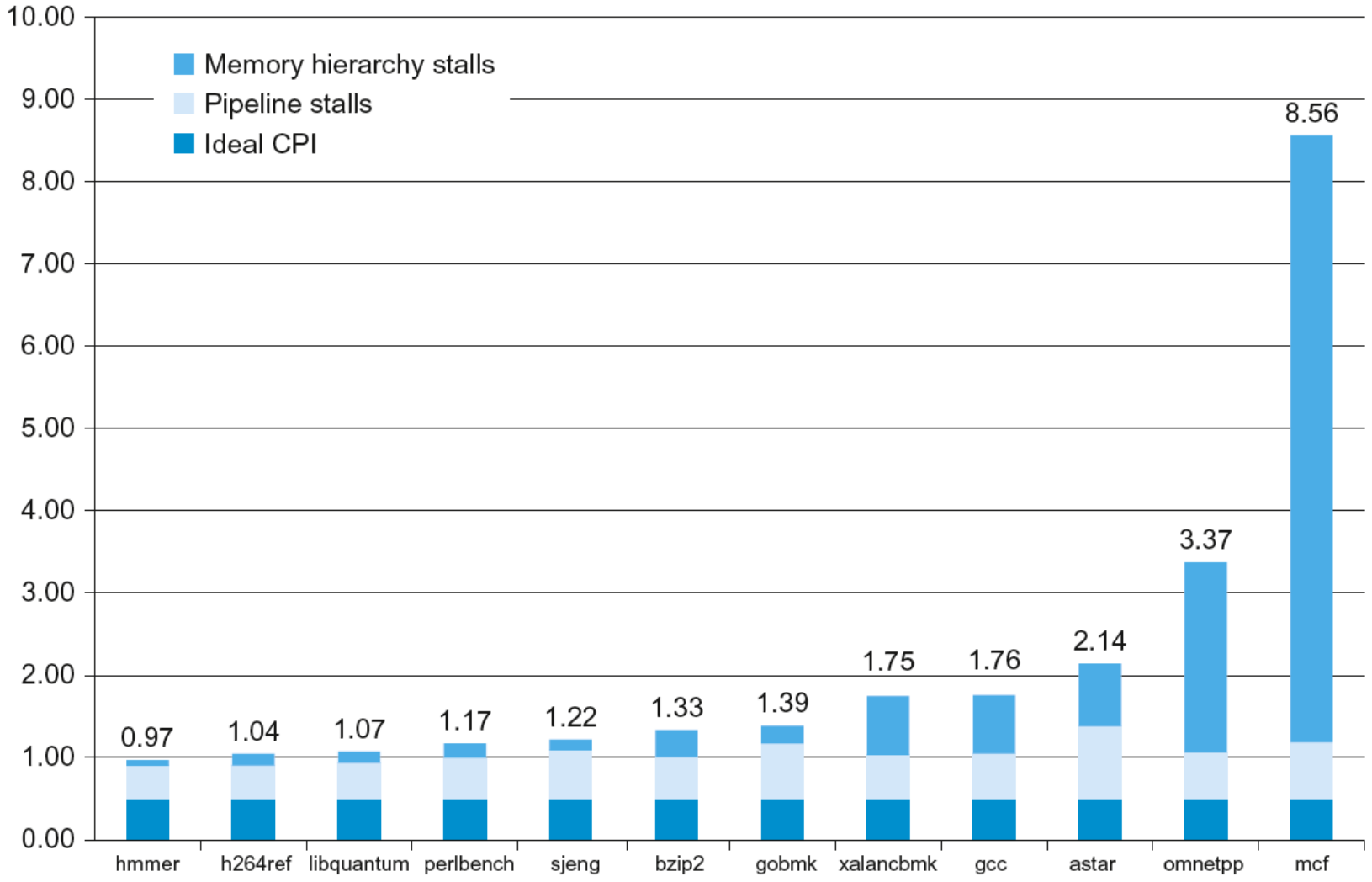
Cortex A53 and Intel i7

Processor	ARM A53	Intel Core i7 920
Market	Personal Mobile Device	Server, cloud
Thermal design power	100 milliWatts (1 core @ 1 GHz)	130 Watts
Clock rate	1.5 GHz	2.66 GHz
Cores/Chip	4 (configurable)	4
Floating point?	Yes	Yes
Multiple issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2	4
Pipeline stages	8	14
Pipeline schedule	Static in-order	Dynamic out-of-order with speculation
Branch prediction	Hybrid	2-level
1st level caches/core	16-64 KiB I, 16-64 KiB D	32 KiB I, 32 KiB D
2nd level caches/core	128-2048 KiB	256 KiB (per core)
3rd level caches (shared)	(platform dependent)	2-8 MB

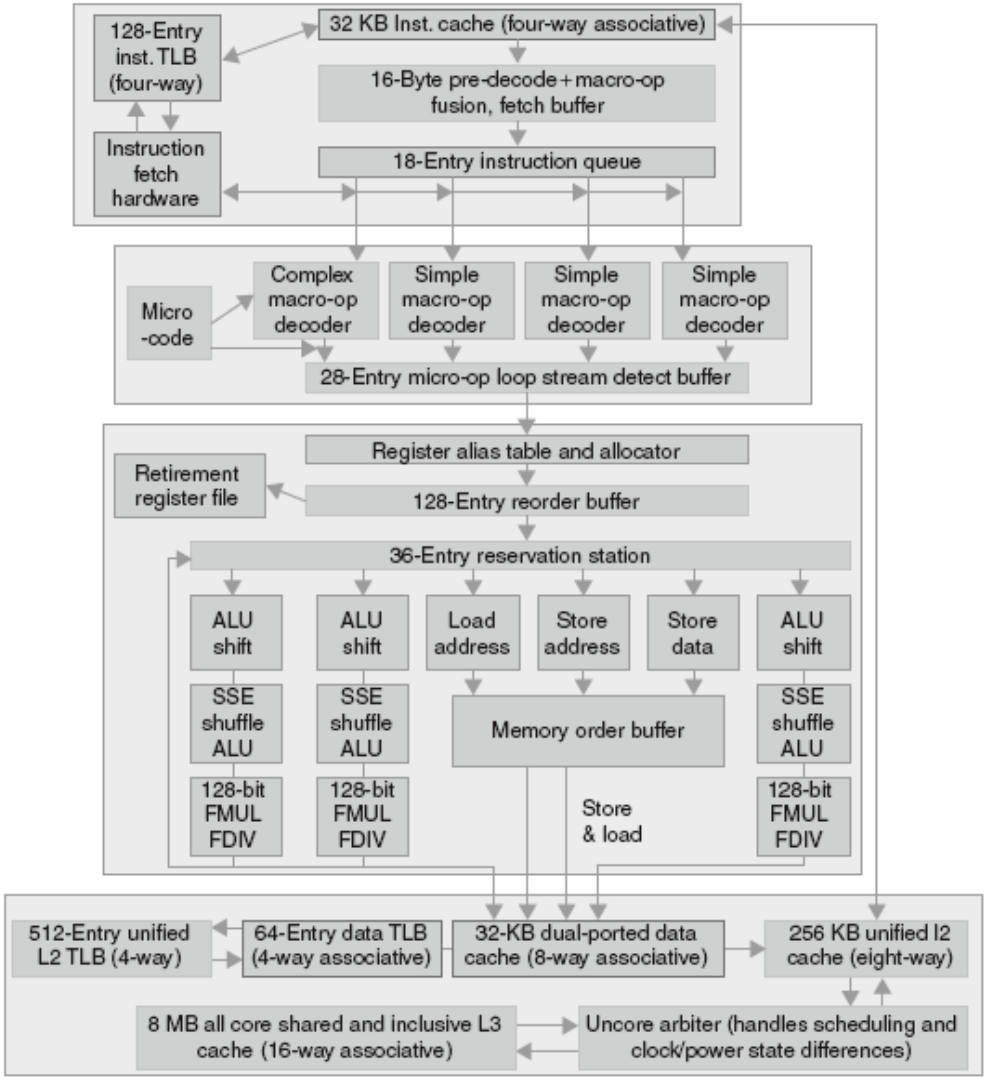
ARM Cortex-A53 Pipeline



ARM Cortex-A53 Performance



Core i7 Pipeline



Core i7 Performance