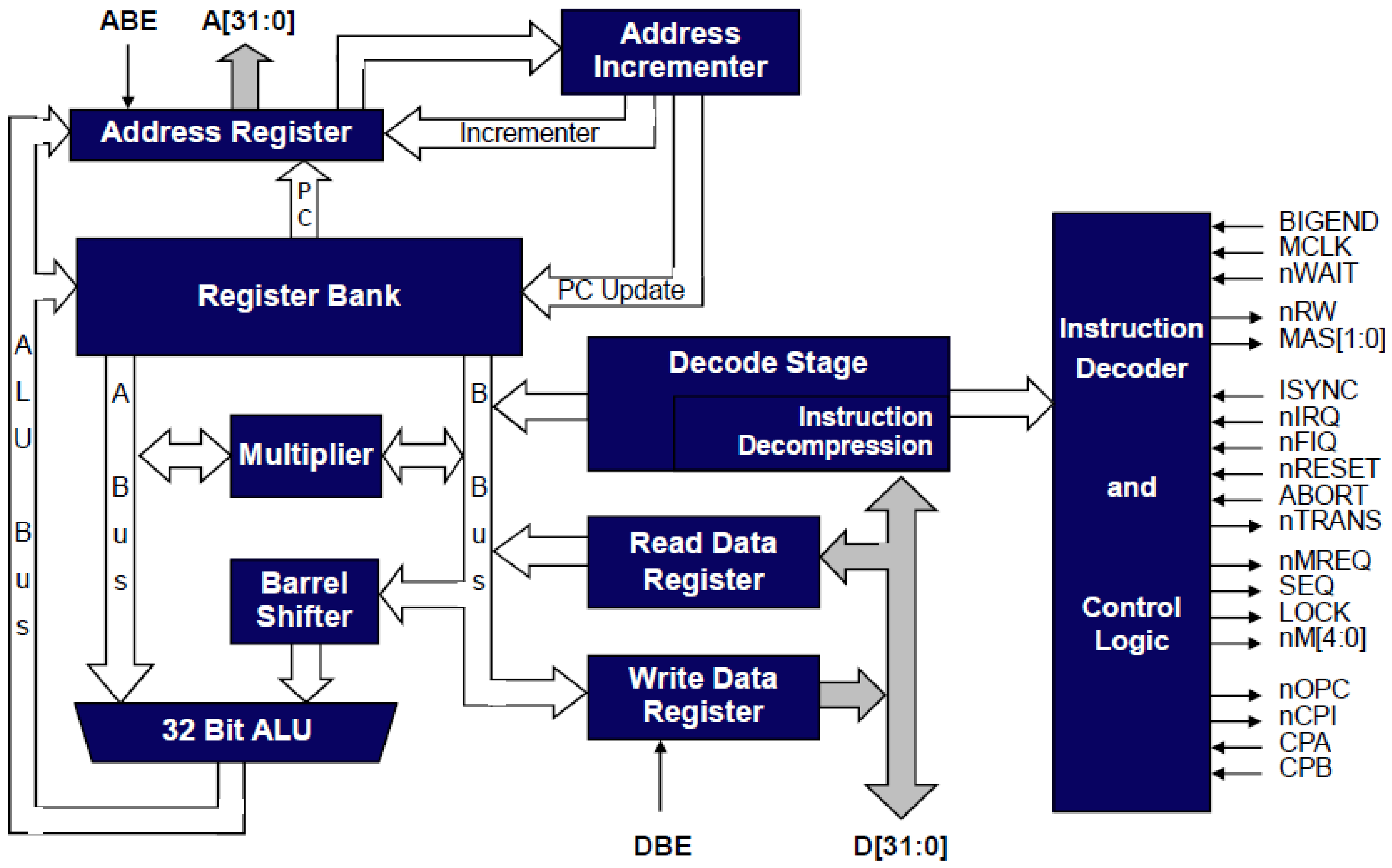
Elements of CPU performance

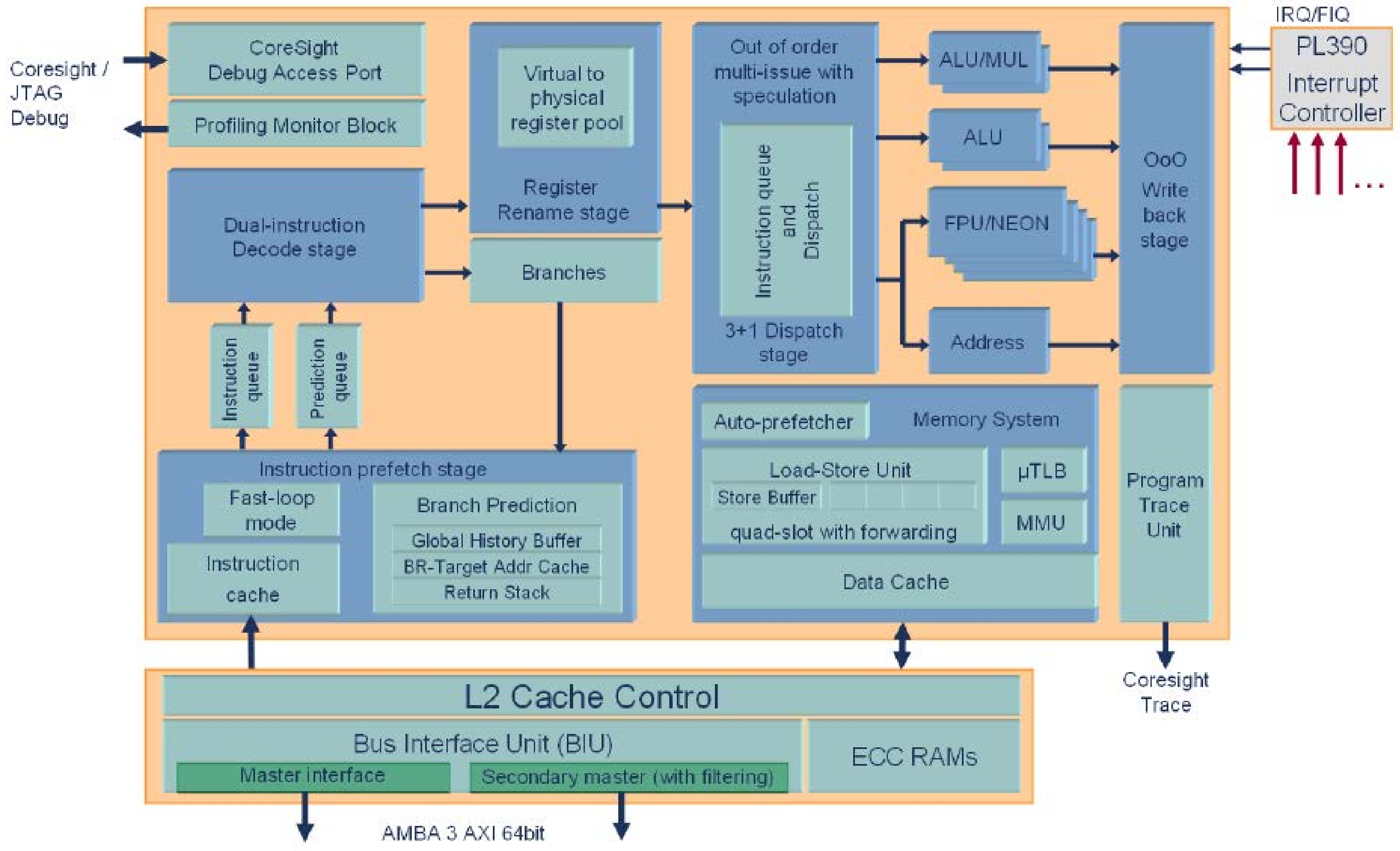
- Cycle time.
- CPU pipeline.
- Superscalar design.
- Memory system.

$$Texec = \left(\frac{instructions}{program}\right)\left(\frac{cycles}{instruction}\right)\left(\frac{\sec onds}{cycle}\right)$$

ARM7TDM CPU Core

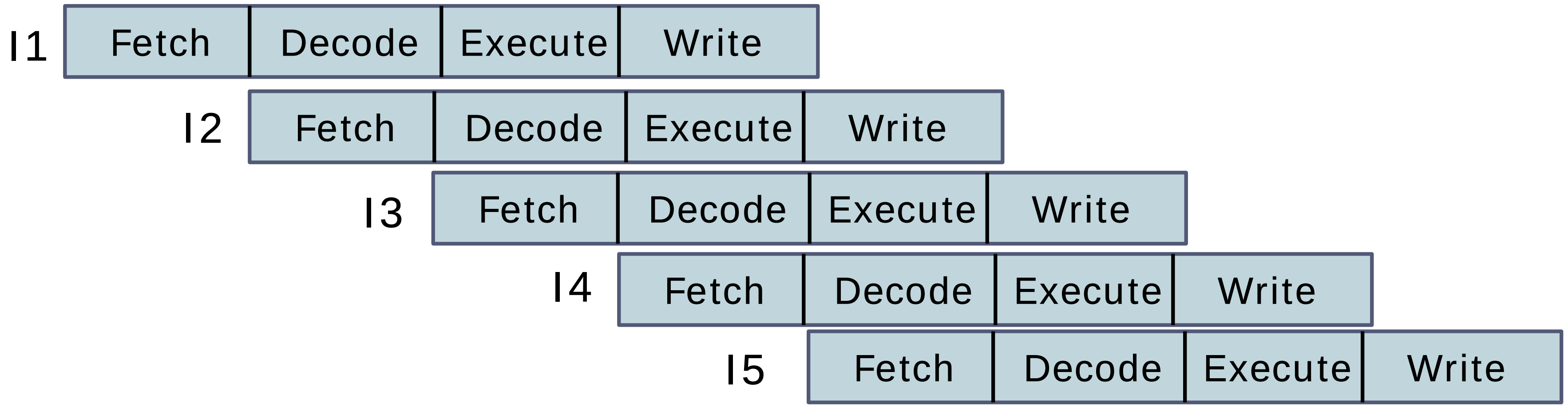


ARM Cortex A-9 Microarchitecture



Pipelining

- Several instructions are executed simultaneously at different stages of completion.

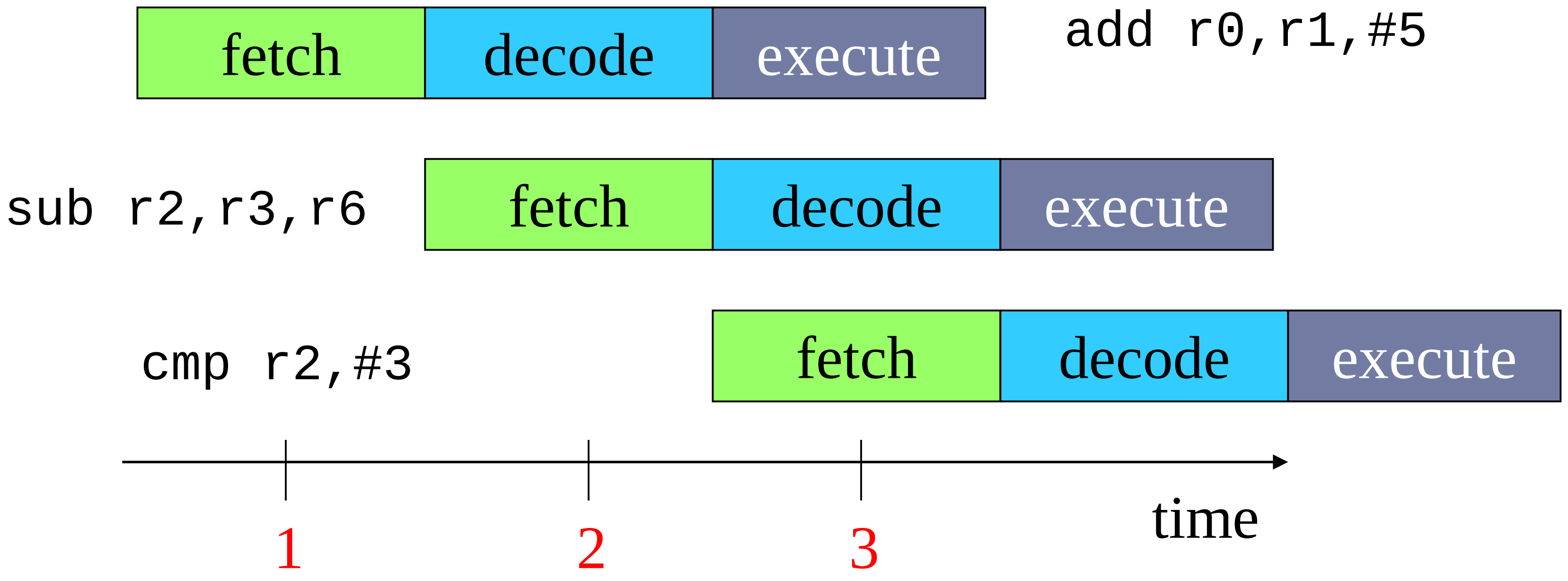


- Various conditions can cause pipeline bubbles that reduce utilization:
  - branches;
  - memory system delays;
  - etc.

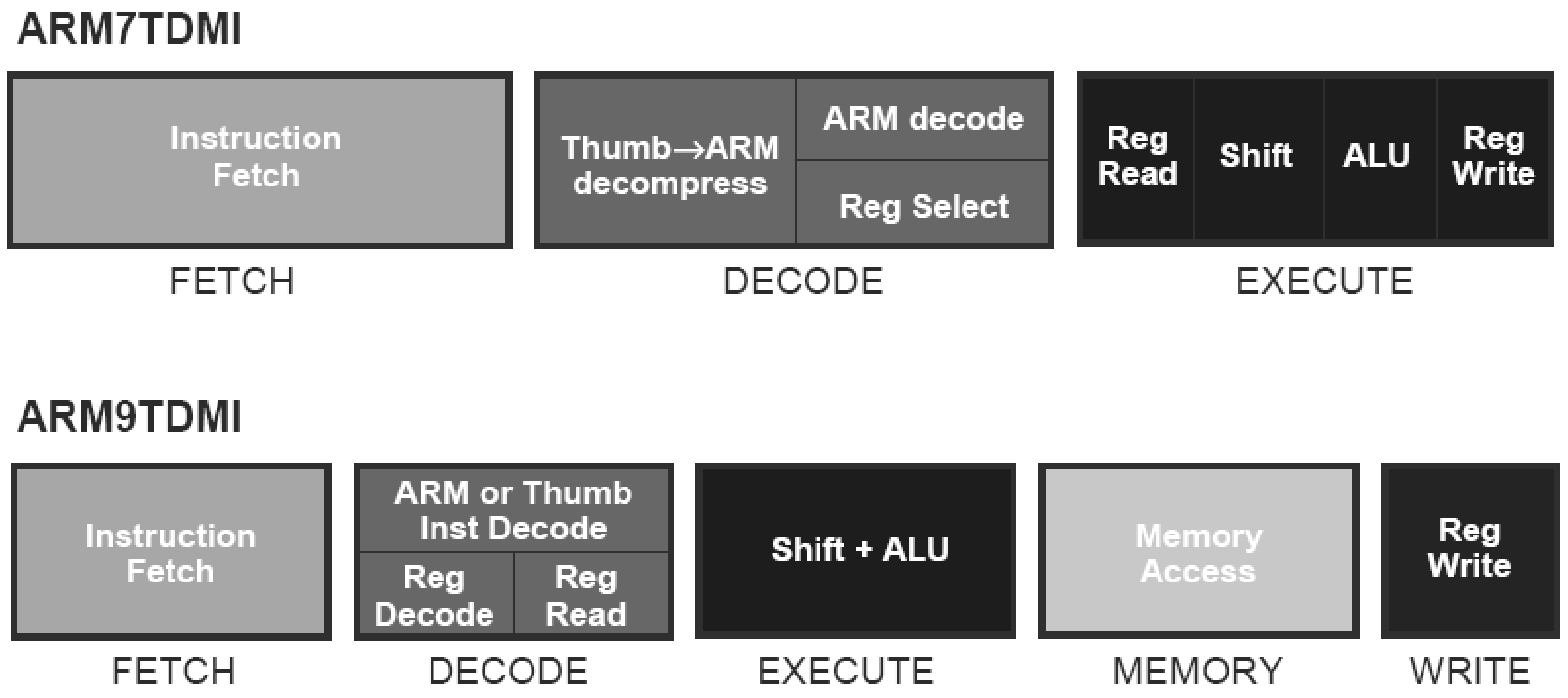
ARM pipeline execution

ARM7 has 3-stage pipes:

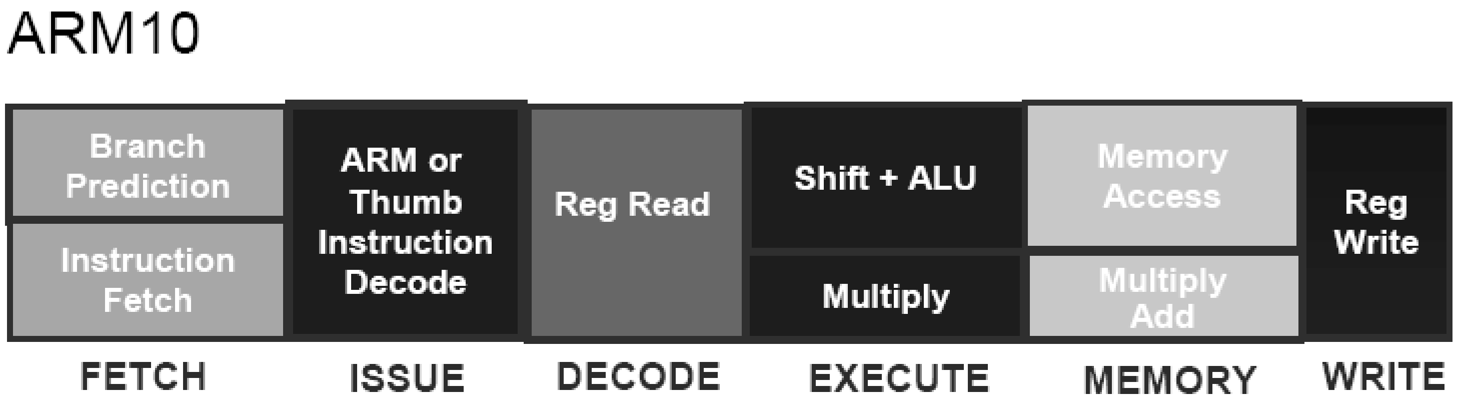
- fetch instruction from memory;
- decode opcode and operands;
- execute.

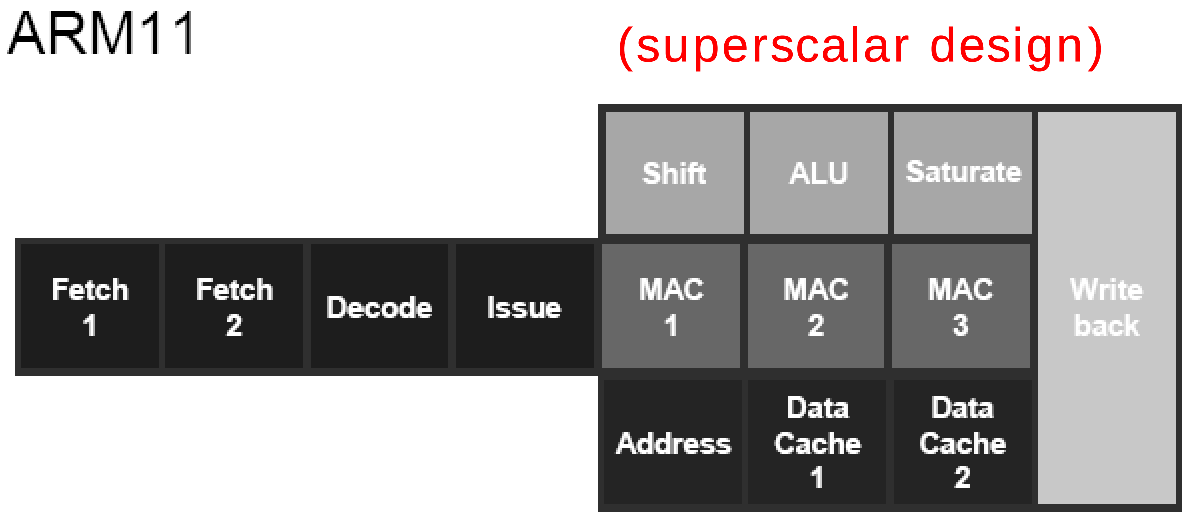


Pipeline changes for ARM9TDMI



ARM10 and ARM11 pipelines





Performance measures

- Latency: time it takes for an instruction to get through the pipeline.
- Throughput: number of instructions executed per time period.

- Pipelining increases throughput without reducing latency.

Assume a program with N, K-stage instructions

Without pipeline: Texec = N*K

With K-stage pipeline: Texec = K + (N-1)

K cycles for 1st instruction

1 cycle to complete each additional instruction

$$\text{Speedup} = \frac{N \times K}{K + (N-1)}$$

For large N

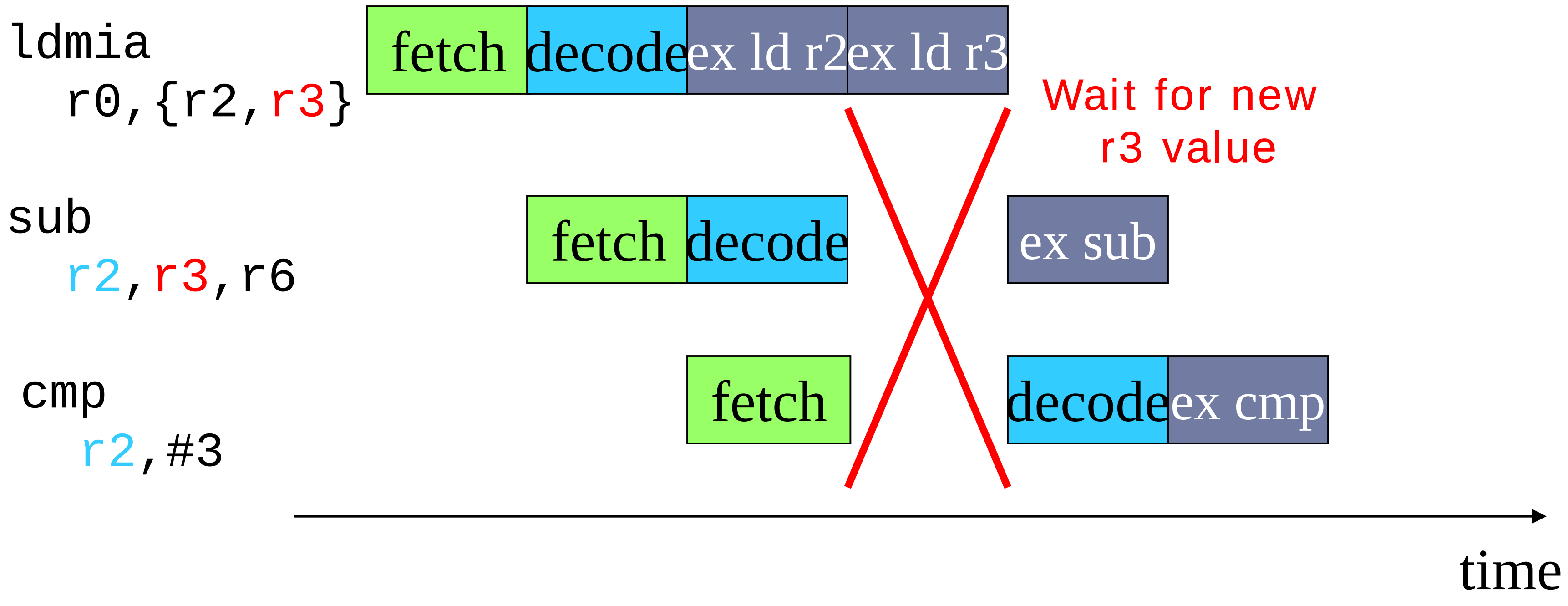
$$Speedup \approx K$$

This assumes no pipeline stalls.

Pipeline stalls

- If every step cannot be completed in the same amount of time, pipeline stalls.
- Bubbles introduced by stall increase latency, reduce throughput.

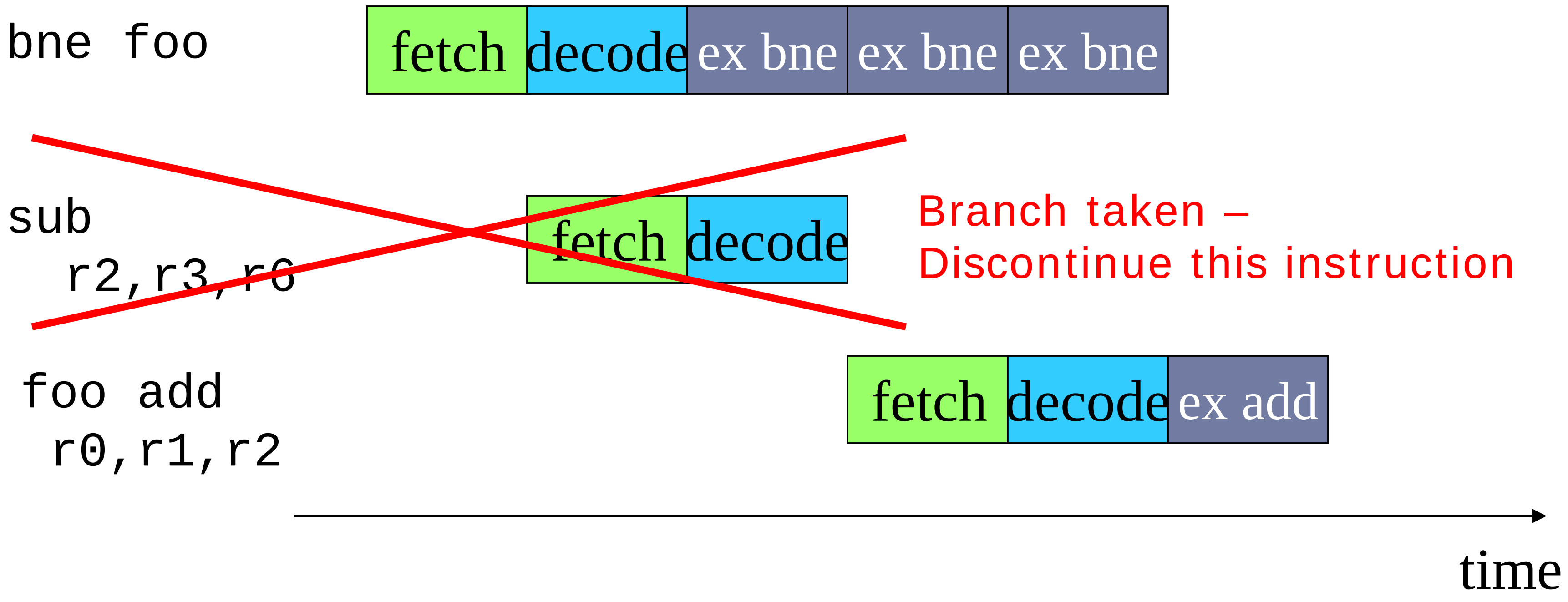
ARM multi-cycle LDMIA instruction



Control stalls

- Branches often introduce stalls (branch penalty).
  - Stall time may depend on whether branch is taken.
- May have to squash instructions that already started executing.
- Don't know what to fetch until condition is evaluated.

ARM pipelined branch



Delayed branch

- To increase pipeline efficiency, delayed branch mechanism requires n instructions after branch always executed whether branch is executed or not.
- SHARC supports delayed and non-delayed branches.
  - Specified by bit in branch instruction.
  - 2 instruction branch delay slot.

Example: ARM execution time

- Determine execution time of FIR filter:

```
for (i=0; i<N; i++)
  f = f + c[i]*x[i];
```

- Only branch in loop test may take more than one cycle.
  - `BLT loop` takes 1 cycle best case, 3 worst case.

FIR filter ARM code

```
; loop initiation code
MOV r0,#0 ; use r0 for i, set to 0
MOV r8,#0 ; use a separate index for arrays
ADR r2,N ; get address for N
LDR r1,[r2] ; get value of N
MOV r2,#0 ; use r2 for f, set to 0
ADR r3,c ; load r3 with address of base of c
ADR r5,x ; load r5 with address of base of x
```

```
        ; loop body
loop    LDR r4,[r3,r8] ; get value of c[i]
        LDR r6,[r5,r8] ; get value of x[i]
        MUL r4,r4,r6 ; compute c[i]*x[i]
        ADD r2,r2,r4 ; add into running sum
        ; update loop counter and array index
        ADD r8,r8,#4 ; add one to array index
        ADD r0,r0,#1 ; add 1 to i
        ; test for exit
        CMP r0,r1
        BLT loop  ; if   i < N, continue loop
loopend   ...
```

FIR filter performance by block

Block	Variable	# instructions	# cycles
Initialization	t_{init}	7	7
Body	t_{body}	4	4
Update	t_{update}	2	2
Test	t_{test}	2	[2,4]

$$t_{loop} = t_{init} + N(t_{body} + t_{update}) + (N-1)\, \underbrace{t_{test,worst}}_{\text{Loop test succeeds is worst case}} + \underbrace{t_{test,best}}_{\text{Loop test fails is best case}}$$

FIR performance on ARM

$$t_{loop} = \underset{7}{t_{init}} + N(\underset{4}{t_{body}} + \underset{2}{t_{update}}) + (N-1)\underset{4}{t_{test,worst}} + \underset{2}{t_{test,best}}$$

N = # times loop executed

$$t_{loop} = 5 + (N \times 10) cycles$$

Superscalar execution

- Superscalar processor can execute several instructions per cycle.
  - Uses multiple pipelined data paths.
- Programs execute faster, but it is harder to determine how much faster.
- Multicore module has multiple processors, each executing separate program "threads"

Data dependencies

- Execution time depends on operands, not just opcode.
- Superscalar CPU checks data dependencies dynamically:

data dependency

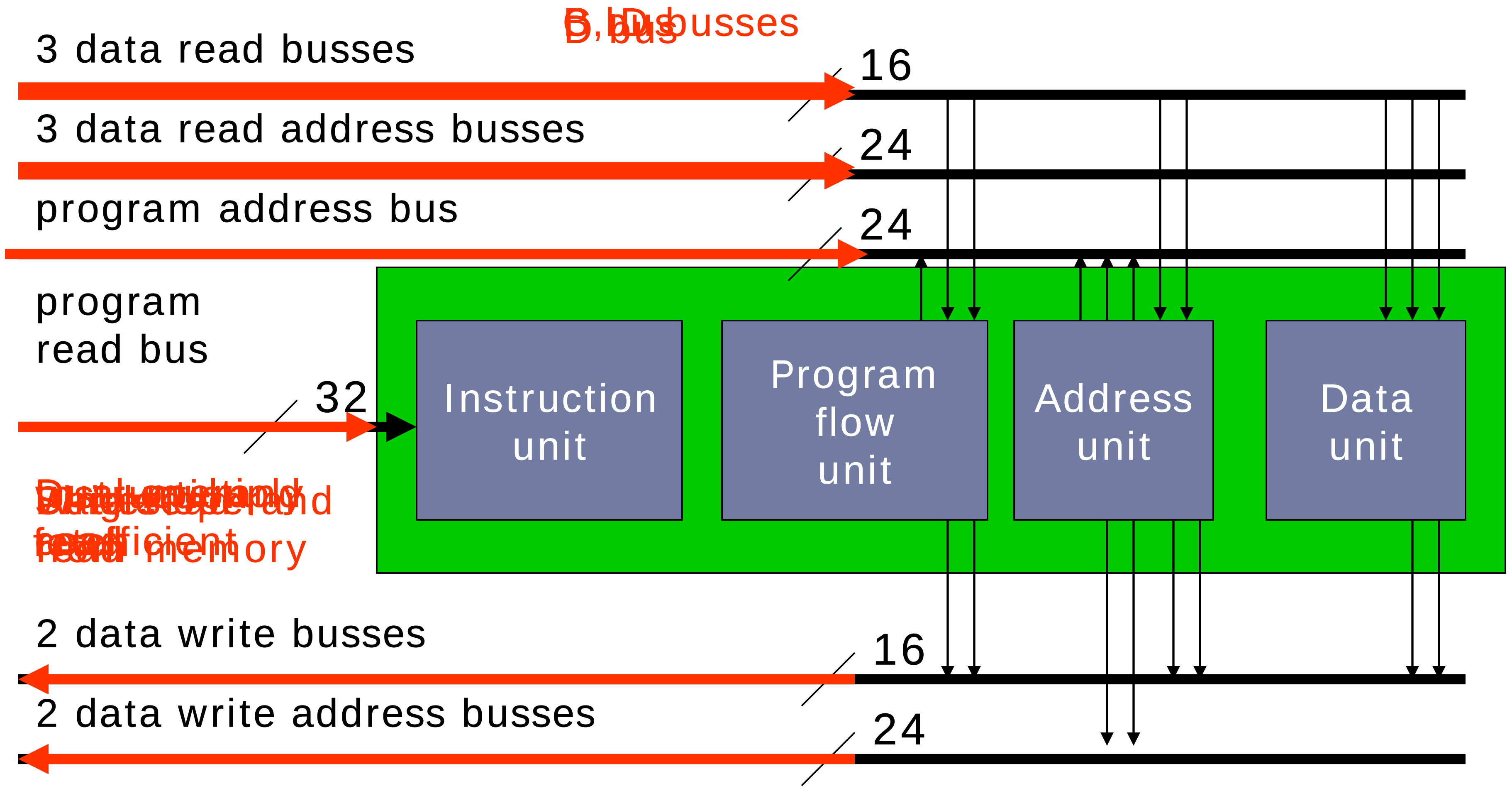
```
add r2,r0,r1
add r3,r2,r5
```

$$\begin{tikzcd}
r0 \arrow[dr] & & r1 \arrow[dl] & \\
& r2 \arrow[dr] & & r5 \arrow[dl] \\
& & r3 &
\end{tikzcd}$$

C55x pipeline

- C55x has 7-stage pipe:
  - fetch;
  - decode;
  - address: computes data/branch addresses;
  - access 1: reads data;
  - access 2: finishes data read;
  - Read stage: puts operands on internal busses;
  - execute.

C55x organization



C55x pipeline hazards

- Processor structure:
  - Three computation units.
  - 14 operators.
- Can perform two operations per instruction.
- Some combinations of operators are not legal.

C55x hazards

- A-unit ALU/A-unit ALU.
- A-unit swap/A-unit swap.
- D-unit ALU,shifter,MAC/D-unit ALU,shifter,MAC
- D-unit shifter/D-unit shift, store
- D-unit shift, store/D-unit shift, store
- D-unit swap/D-unit swap
- P-unit control/P-unit control