ELE 655
Microprocessor System Design

Section 2 – Instruction Level Parallelism

Class 5 – Multiple Issue

ILP

Instruction Level Parallelism

- How do we achieve a CPI < 1
  - Superscalar architecture supports multiple execution units
  - Pipeline make for efficient operation
  - Need to have some way to complete more than 1 instruction per clock cycle
  - → need to issue more than 1 instruction per clock cycle

ILP

Instruction Level Parallelism

- 3 possible solutions
  - Statically scheduled superscalar processors
  - VLIW (very long instruction word) processors
  - dynamically scheduled superscalar processors

ILP

Instruction Level Parallelism

- 3 possible solutions

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the ARM Coretex A8
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

ILP

Instruction Level Parallelism

- VLIW
  - Package multiple operations into one instruction
  - Static scheduling
  - Example VLIW processor:
    - One integer instruction (or branch)
    - Two independent floating-point operations
    - Two independent memory references
  - Must be enough parallelism in code to fill the available slots
  - **Fig 3.16**

ILP
Instruction Level Parallelism

- VLIW
  - Disadvantages
    - Statically finding parallelism
      - May be difficult to find this in a basic block
      - Global scheduling algorithms
    - Code size
      - Need to do a lot of loop unrolling → larger code size
      - Unused slots create wasted memory space (assuming aligned words)
    - No hazard detection hardware
      - Functional units forced to work in lockstep
    - Binary code compatibility
      - Instructions are tightly coupled to both instruction set and implementation (numbers and types of functional units)

ILP
Instruction Level Parallelism

- Current processors
  - Dynamically scheduled with speculation

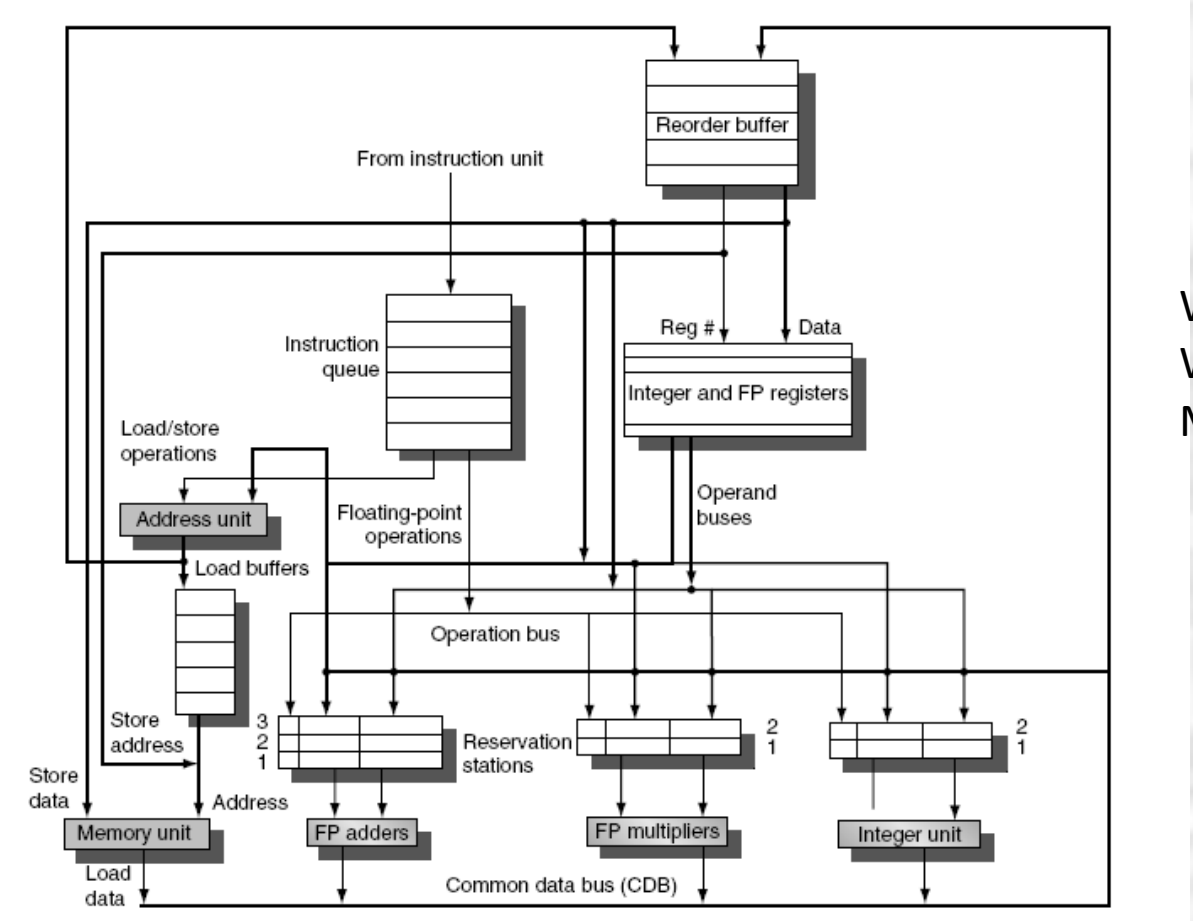
    add Multiple issue

  - Simple 2 instruction issue
    - Operate on half clock cycles
  - More complex multiple issues requires
    - Complex issue logic
    - Wider buses to handle multiple transactions per cycle
    - Ability to handle dependencies between instructions issued together
      - Need to update tables in parallel

ILP

Instruction Level Parallelism

- Current processors



Wider CDB
Wider Operand bus
More complex issue logic

ILP
Instruction Level Parallelism

- Multiple Issue Process
  - Given a bundle of instructions to issue
    - Limit the number of each type of instruction allowed
    - Anything over the limit is broken out of the bundle and waits
    - E.g.. 2 L/S, 3 Add, 1 FP
  - Once the bundle is limited (but before the specific instruction is known)
    - Assign a ROB to each instruction
    - Assign a RS for each execution unit to the bundle
    - If either are limited – break the bundle
  - Analyze the dependencies between instructions in the bundle
  - Update the ROB and RS based on dependencies

ILP

Instruction Level Parallelism

- Multiple Issue Process – no speculation

Iteration number	Instructions		Issues at clock cycle number	Executes at clock cycle number	Memory access at clock cycle number	Write CDB at clock cycle number	Comment
1	LD	R2,0(R1)	1	2	3	4	First issue
1	DADDIU	R2,R2,#1	1	5		6	Wait for LW
1	SD	R2,0(R1)	2	3	7		Wait for DADDIU
1	DADDIU	R1,R1,#8	2	3		4	Execute directly
1	BNE	R2,R3,LOOP	3	7			Wait for DADDIU
2	LD	R2,0(R1)	4	8	9	10	Wait for BNE
2	DADDIU	R2,R2,#1	4	11		12	Wait for LW
2	SD	R2,0(R1)	5	9	13		Wait for DADDIU
2	DADDIU	R1,R1,#8	5	8		9	Wait for BNE
2	BNE	R2,R3,LOOP	6	13			Wait for DADDIU
3	LD	R2,0(R1)	7	14	15	16	Wait for BNE
3	DADDIU	R2,R2,#1	7	17		18	Wait for LW
3	SD	R2,0(R1)	8	15	19		Wait for DADDIU
3	DADDIU	R1,R1,#8	8	14		15	Wait for BNE
3	BNE	R2,R3,LOOP	9	19			Wait for DADDIU

ILP

Instruction Level Parallelism

- Multiple Issue Process – with speculation

Iteration number	Instructions	Issues at clock number	Executes at clock number	Read access at clock number	Write CDB at clock number	Commits at clock number	Comment
1	LD R2,0(R1)	1	2	3	4	5	First issue
1	DADDIU R2,R2,#1	1	5		6	7	Wait for LW
1	SD R2,0(R1)	2	3			7	Wait for DADDIU
1	DADDIU R1,R1,#8	2	3		4	8	Commit in order
1	BNE R2,R3,LOOP	3	7			8	Wait for DADDIU
2	LD R2,0(R1)	4	5	6	7	9	No execute delay
2	DADDIU R2,R2,#1	4	8		9	10	Wait for LW
2	SD R2,0(R1)	5	6			10	Wait for DADDIU
2	DADDIU R1,R1,#8	5	6		7	11	Commit in order
2	BNE R2,R3,LOOP	6	10			11	Wait for DADDIU
3	LD R2,0(R1)	7	8	9	10	12	Earliest possible
3	DADDIU R2,R2,#1	7	11		12	13	Wait for LW
3	SD R2,0(R1)	8	9			13	Wait for DADDIU
3	DADDIU R1,R1,#8	8	9		10	14	Executes earlier
3	BNE R2,R3,LOOP	9	13			14	Wait for DADDIU

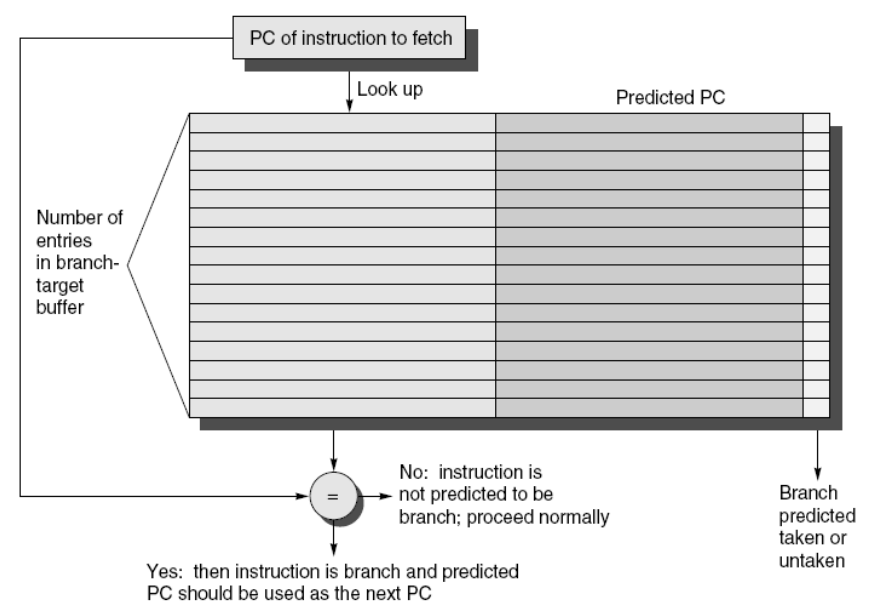
ILP
Instruction Level Parallelism

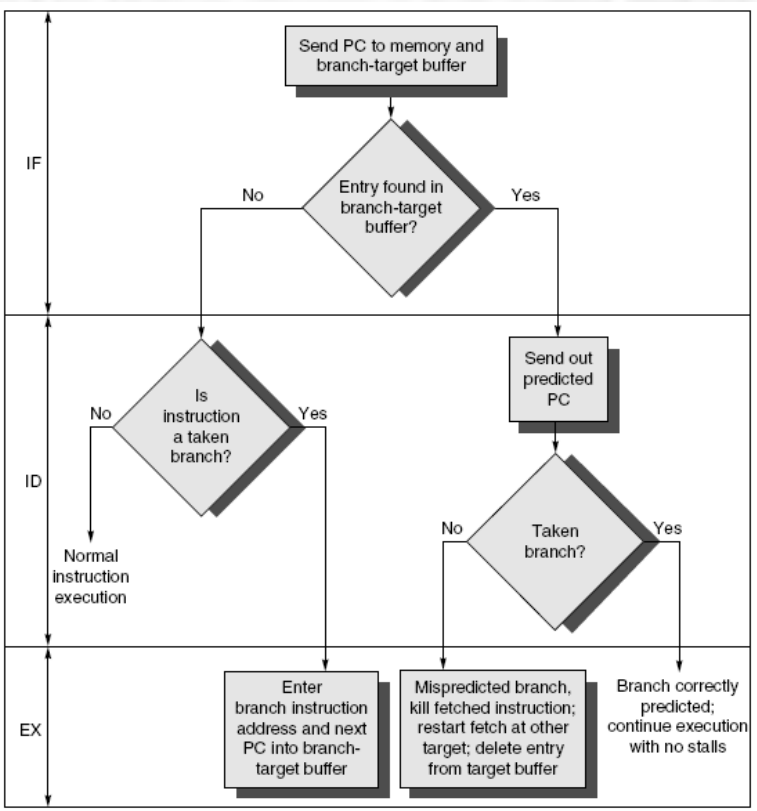
- Increasing Fetch Bandwidth
  - Reduce the delays associated with branch calculations
  - Branch Target Buffer
    - Buffer of branch PC locations for TAKEN branches
    - Includes the PC of the new (taken) location
    - Each instruction is checked against the buffer
      - If not in the buffer
        - Not a branch
        - A not taken branch
      - If in the buffer
        - It is a taken branch and the PC is loaded with the predicted PC
    - → no branch delay

ILP

Instruction Level Parallelism

- Branch Target Buffer
  - Looks like a cache





ILP
Instruction Level Parallelism

- Branch Target Buffer
  - Modification – Branch folding
  - Instead of storing predicted PC, store 1 or more actual instructions from the predicted location
  - Provides the instruction immediately instead of at the next fetch cycle
  - Can lead to 0 cycle branches

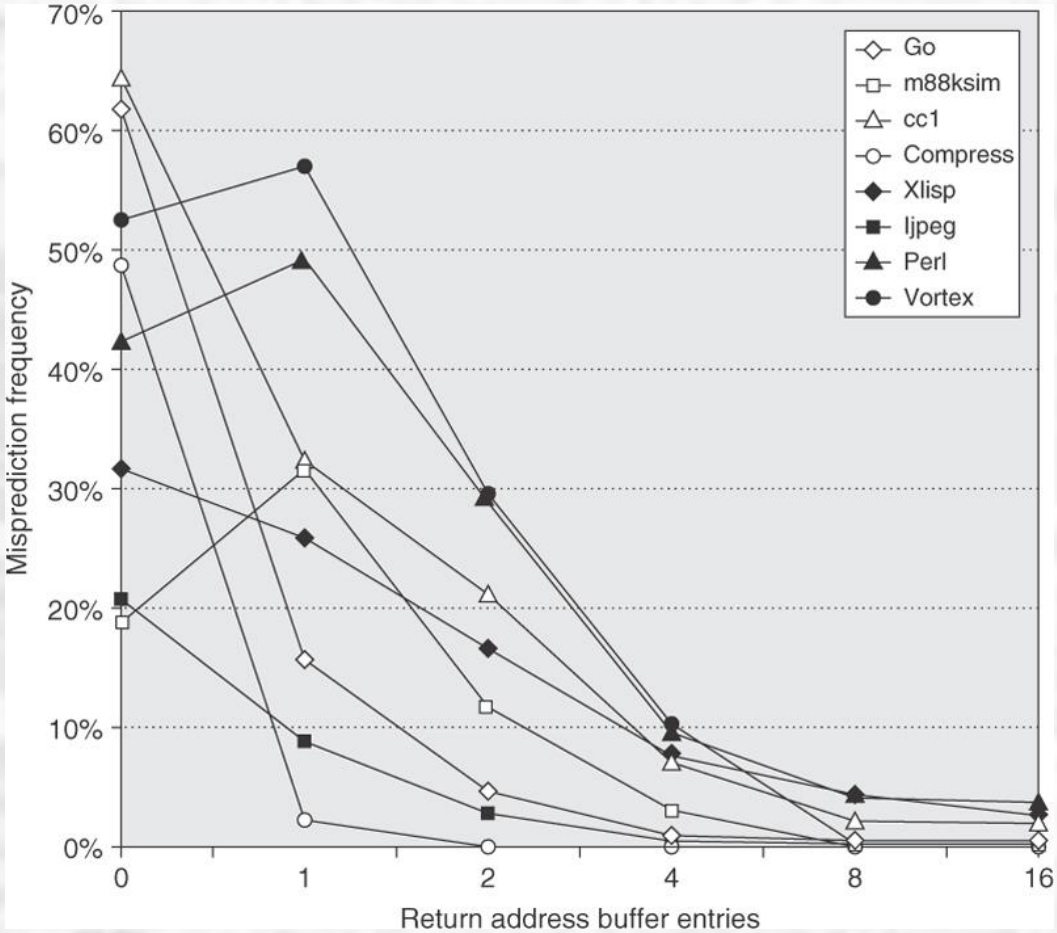
ILP

Instruction Level Parallelism

- Return Address Prediction
  - Procedure returns can be a significant portion of all branches (>15%)
  - These involve indirect jumps (location not known until run time)
  - Since procedures can be called from many locations, the branch target buffer is not very effective
  - Create a small stack that pushes the PC on calls and pops the address on returns
    - If it is sufficiently deep → very accurate performance

ILP
Instruction Level Parallelism

- Return Address Prediction



ILP

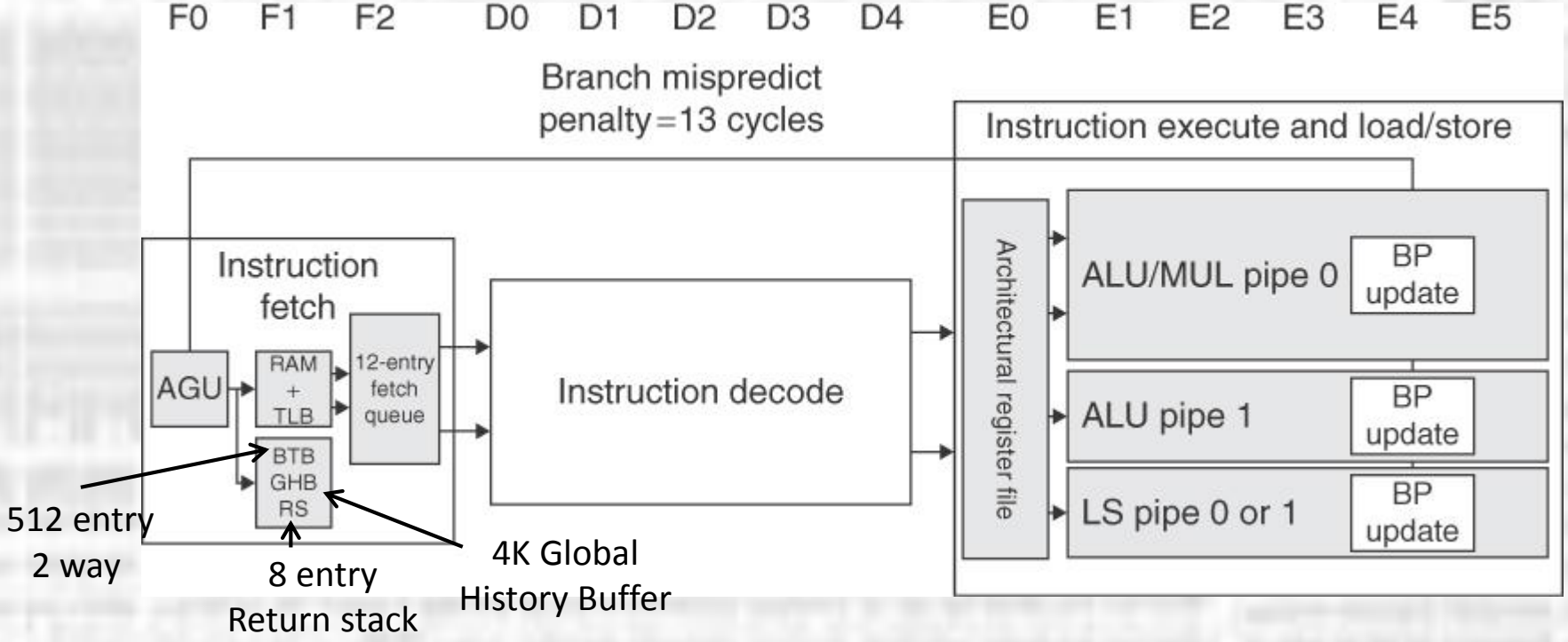
Instruction Level Parallelism

- Multiple Issue
  - Our instruction fetch function has become very complex
  - Break it away from the rest of the pipeline and introduce a buffer between the fetch logic and the pipeline
  - Allows the fetch circuitry to operate while keeping the pipeline fed

ILP

Instruction Level Parallelism

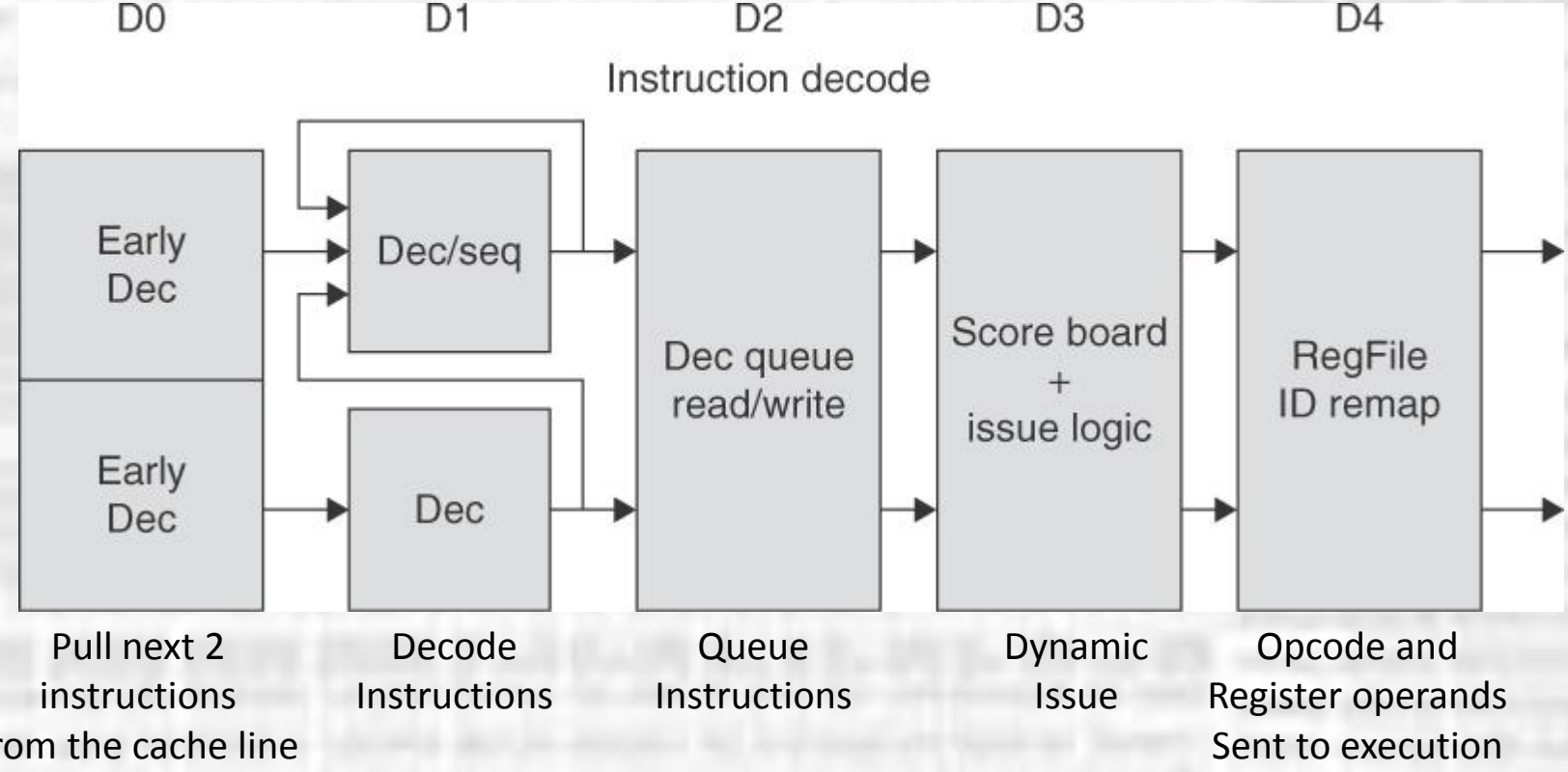
- Arm Cortex A8
  - Dual Issue
  - Statically Scheduled



ILP

Instruction Level Parallelism

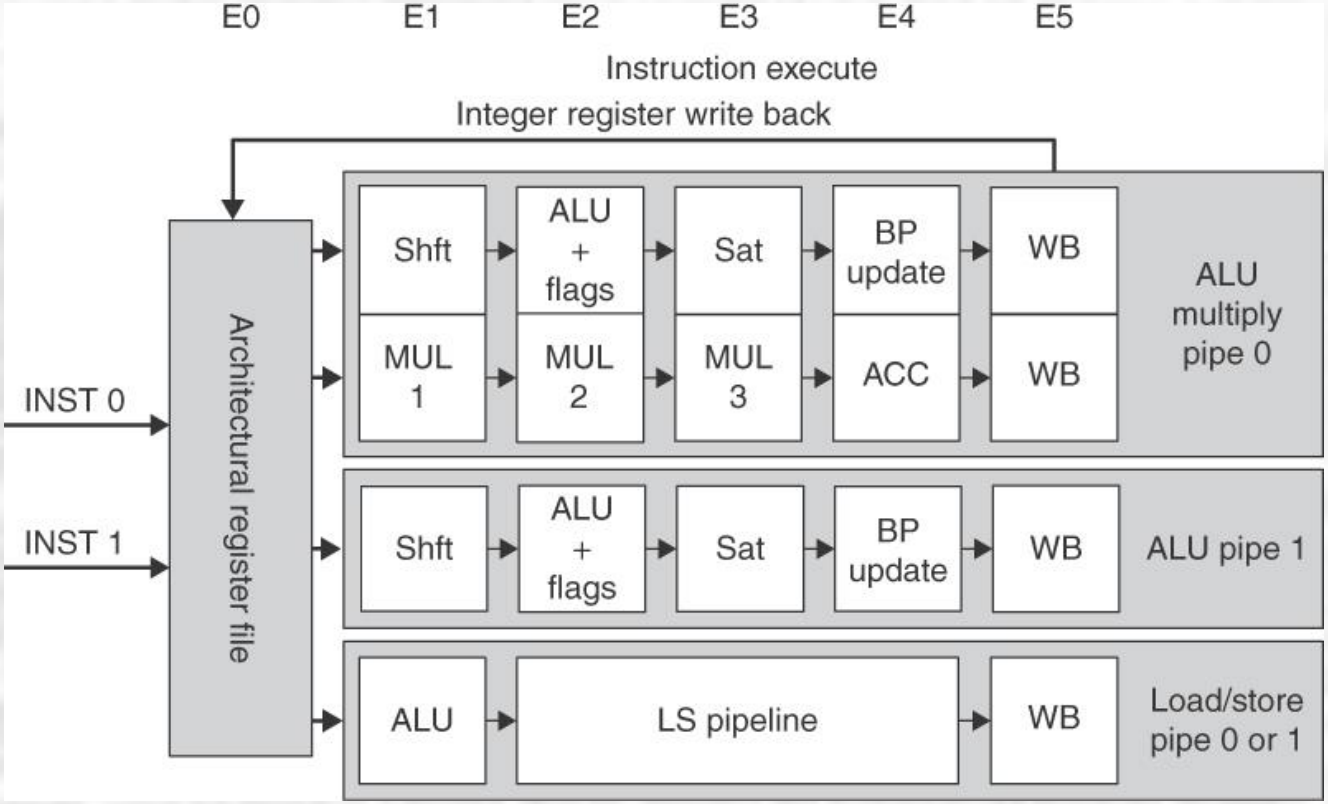
- Arm Cortex A8



ILP

Instruction Level Parallelism

- Arm Cortex A8

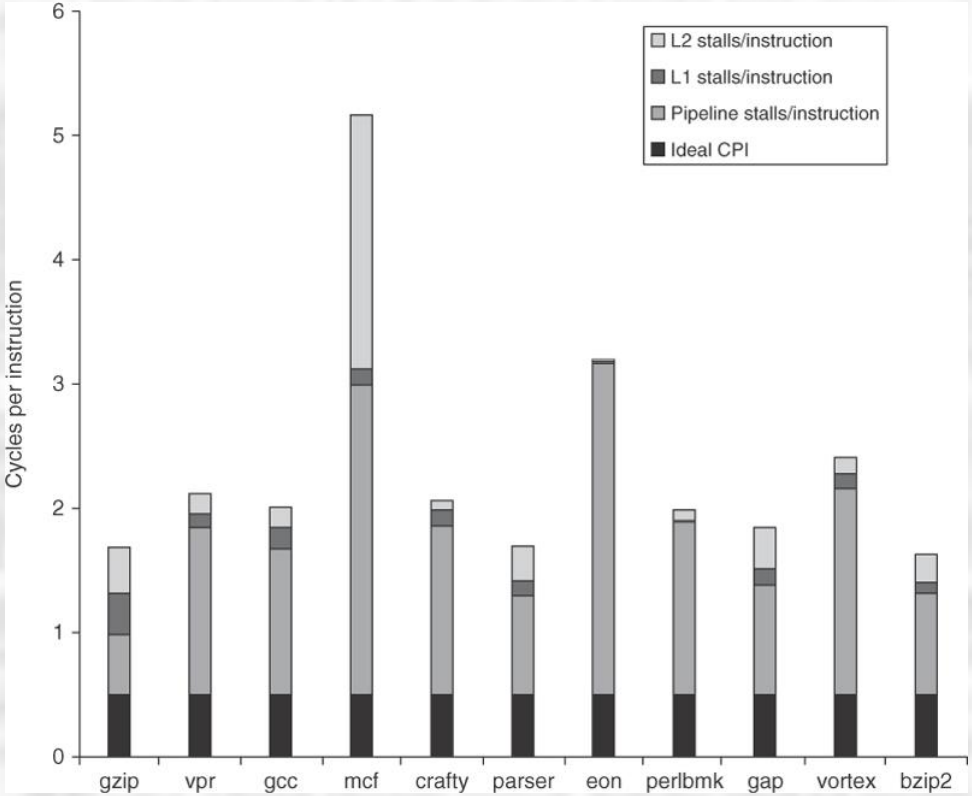


Shared Arithmetic / Multiply Execution pipe

ILP

Instruction Level Parallelism

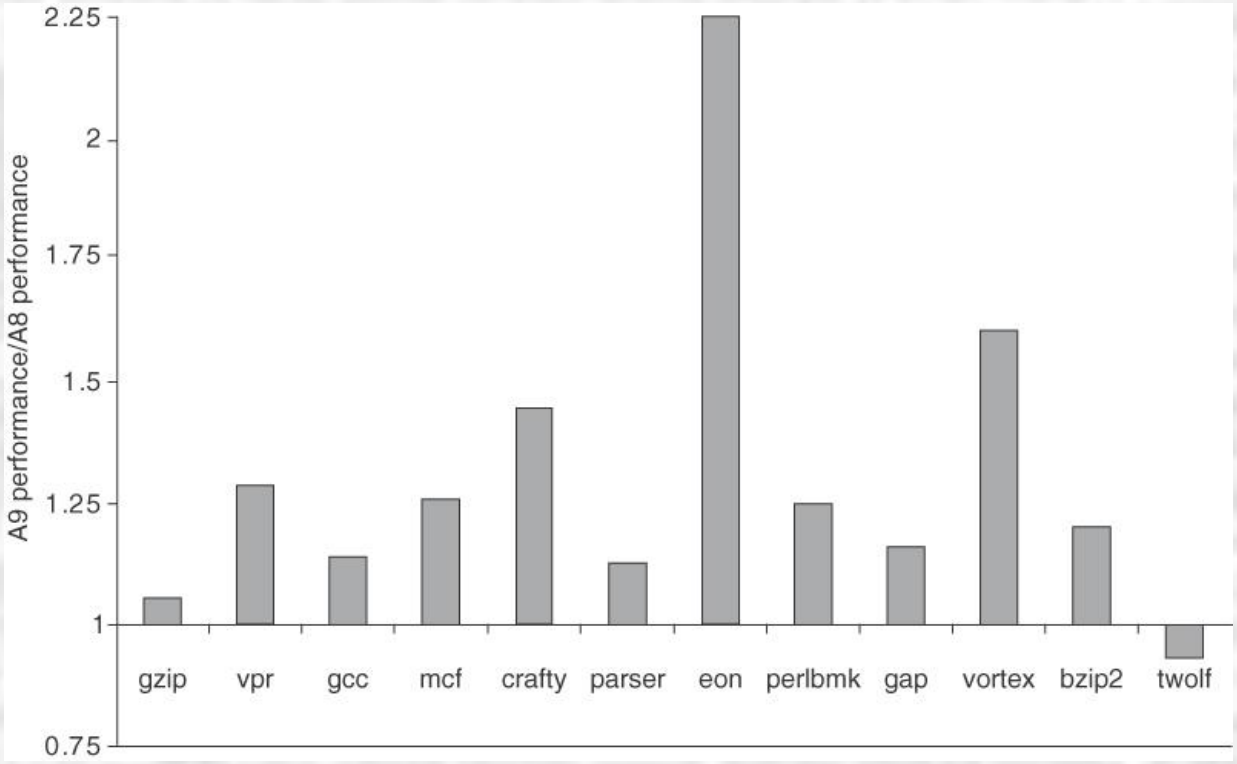
- Arm Cortex A8
  - Ideal CPI = 0.5



ILP

Instruction Level Parallelism

- Arm Cortex A9
  - Dynamically scheduled with speculation
  - Ideal CPI = 0.5



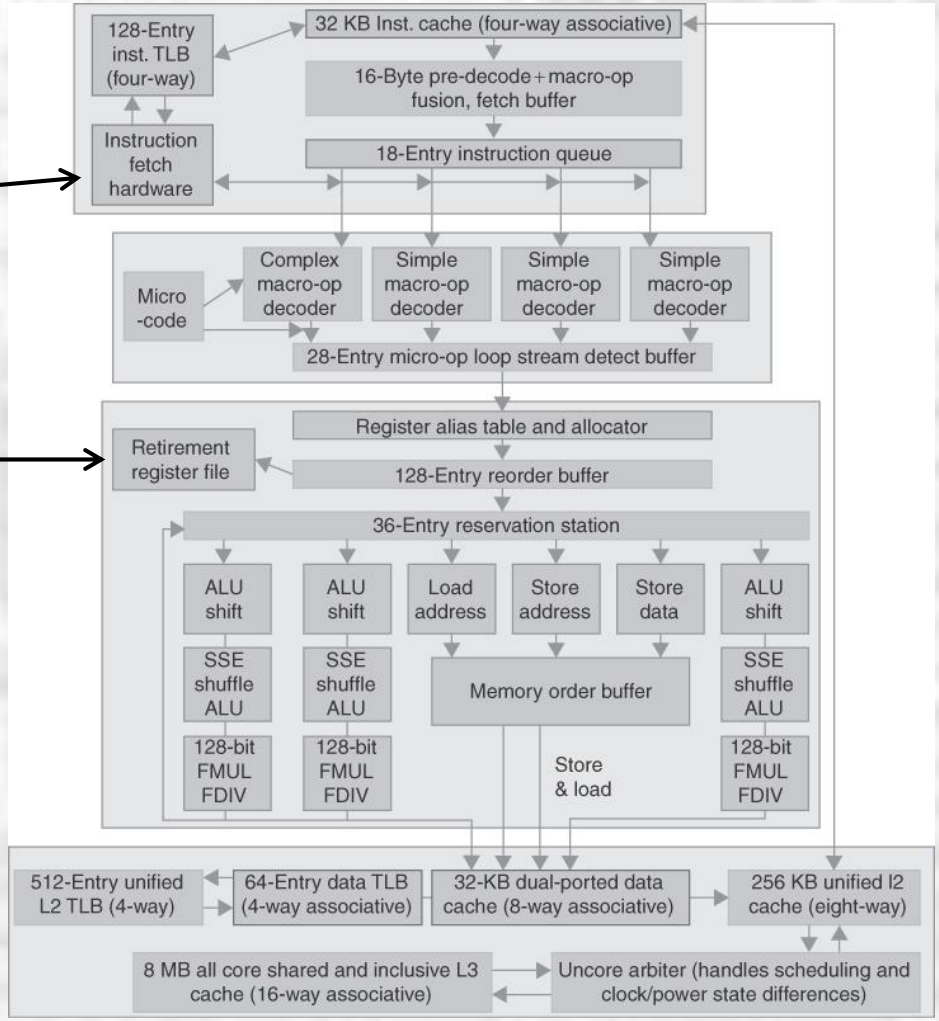
ILP

Instruction Level Parallelism

- Intel Core i7

Multi-level BTB
Return address stack

Update committed register s
In order



ILP

Instruction Level Parallelism

- Intel Core i7