Lecture 9 Addressing and Instruction Level Pipelining

Learning objectives

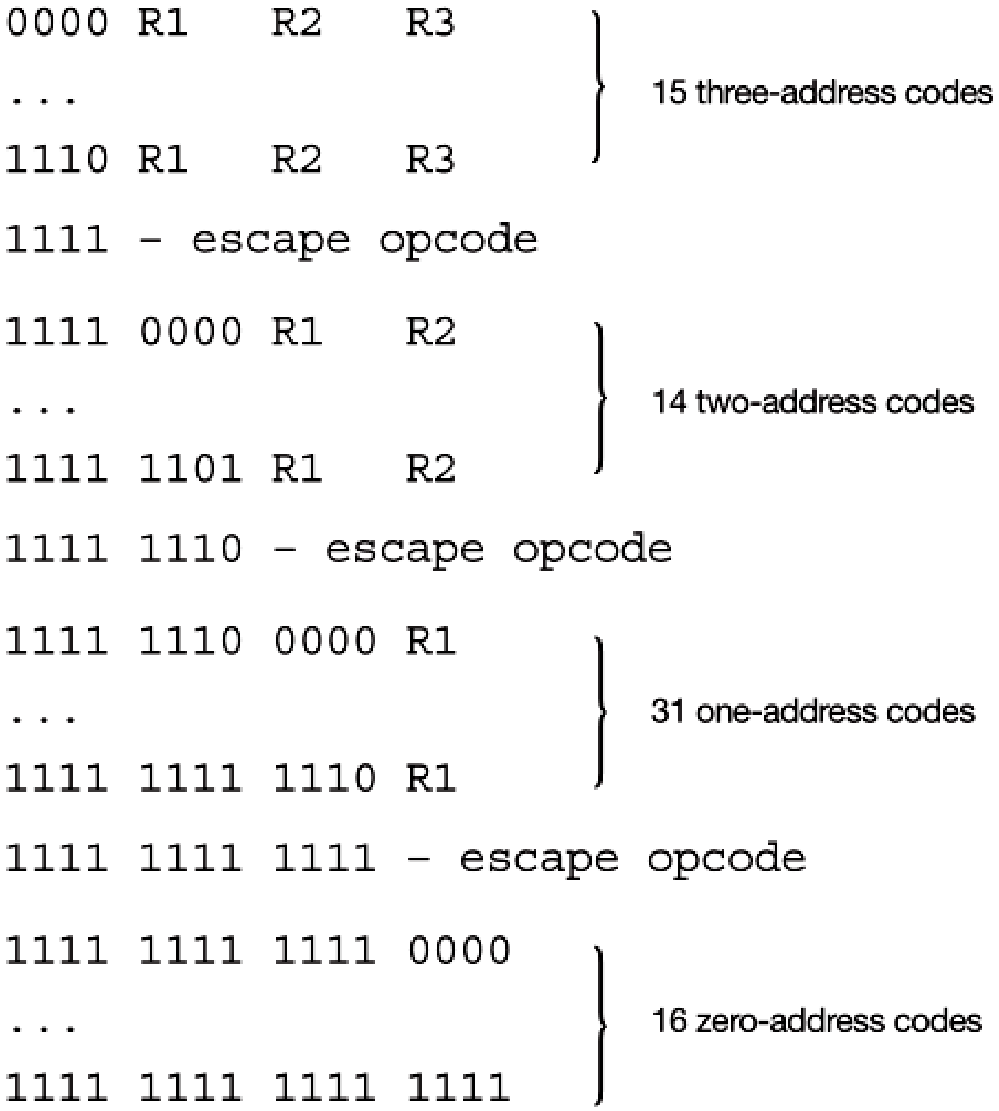
- Addressing modes for operations
- Instruction-level pipelining
- Real-world examples of ISAs

Instruction Formats

- We have seen how instruction length is affected by the number of operands supported by the ISA.
- In any instruction set, not all instructions require the same number of operands.
- Operations that require no operands, such as **`HALT`**, necessarily waste some space when fixed-length instructions are used.
- One way to recover some of this space is to use expanding opcodes.

Instruction Formats

- If we allow the length of the opcode to vary, we could create a very rich instruction set:
  - EX for 16-bit flexible instruction format



```
0000 R1   R2   R3      }
...                    }  15 three-address codes
1110 R1   R2   R3      }
1111 - escape opcode
1111 0000 R1   R2      }
...                    }  14 two-address codes
1111 1101 R1   R2      }
1111 1110 - escape opcode
1111 1110 0000 R1      }
...                    }  31 one-address codes
1111 1111 1110 R1      }
1111 1111 1111 - escape opcode
1111 1111 1111 0000    }
...                    }  16 zero-address codes
1111 1111 1111 1111    }
```

How Many Addresses should it be?

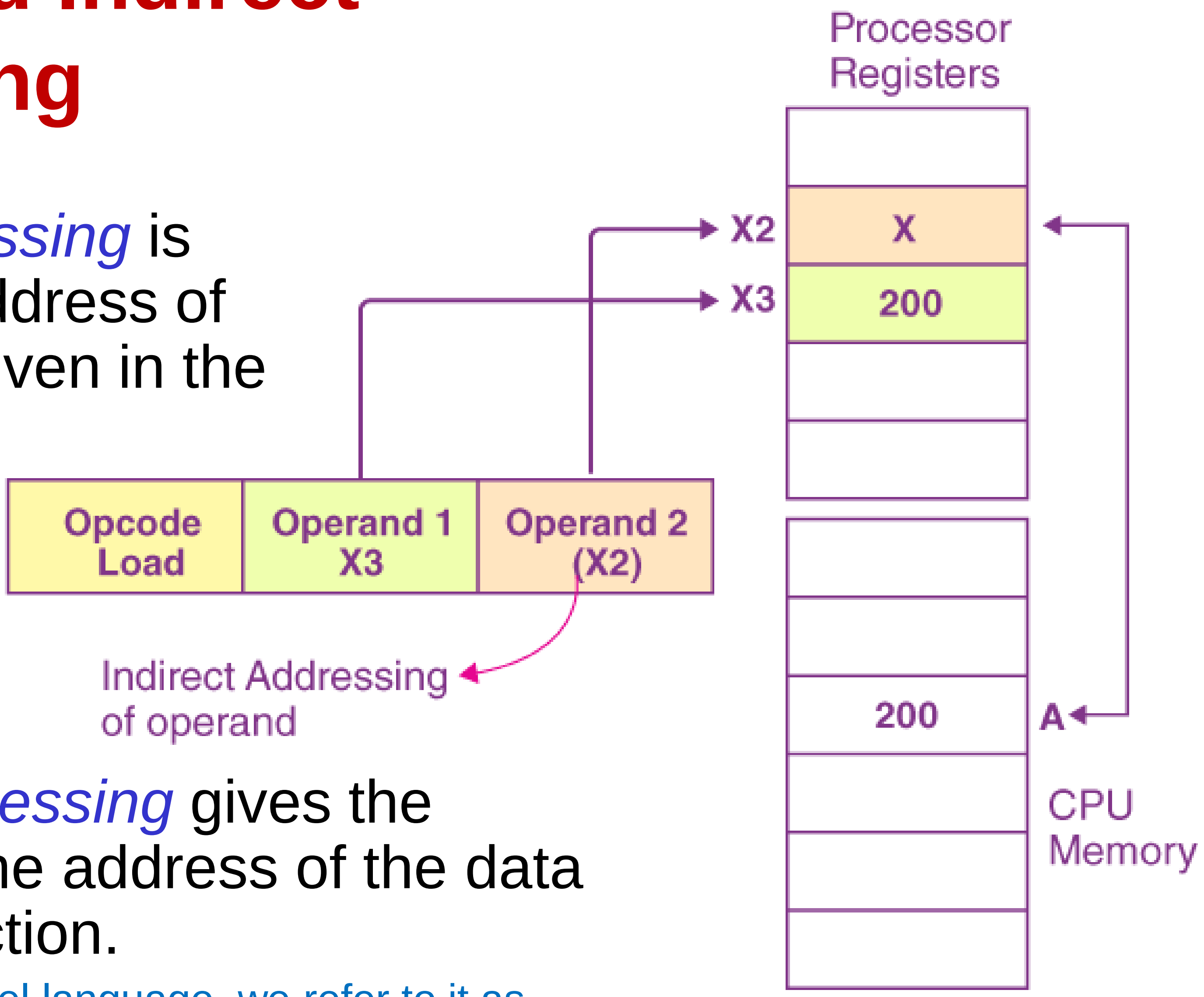
- It is a basic design decision
- More addresses
    - More complex (powerful?) instructions
    - More registers
        - Inter-register operations allow quicker and flexible
    - Fewer instructions per program
- Fewer addresses
    - Less complex instructions
    - More instructions per program
    - Faster fetch/execution of instruction
    - Result in longer execution time, more complex for a program

Addressing mode

- Addressing modes specify where an operand is located.
- They can specify (1) *a constant*, (2) *a register*, or (3) *a memory location*.
- The actual location of an operand is its *effective address (EA)*.
- Certain addressing modes allow us to determine the address of an operand dynamically.

Direct and Indirect Addressing

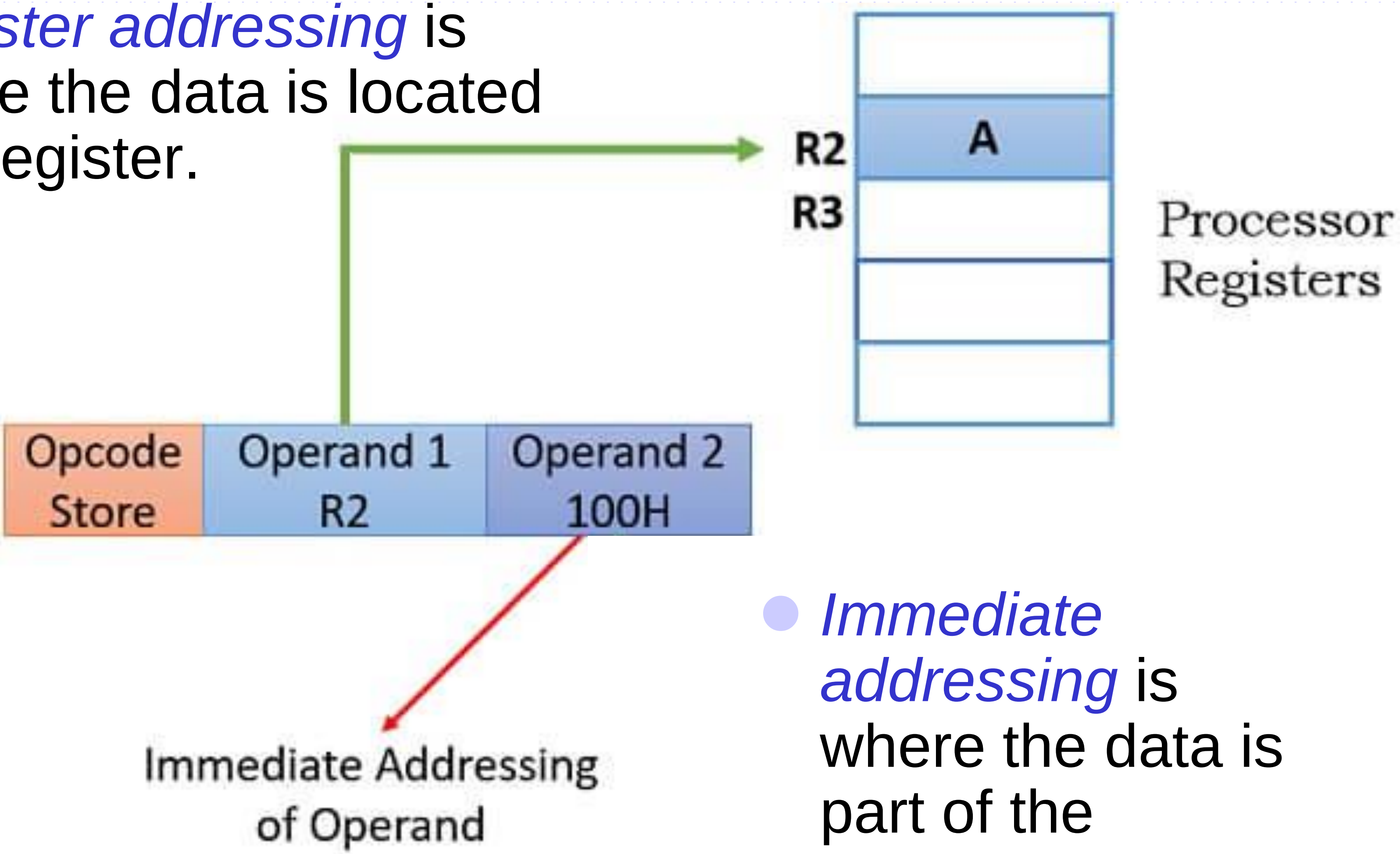
- *Direct addressing* is where the address of the data is given in the instruction.



- *Indirect addressing* gives the address of the address of the data in the instruction.
  - In a higher-level language, we refer to it as pointers.
- ADD X3, X2

Register and Immediate addressing mode

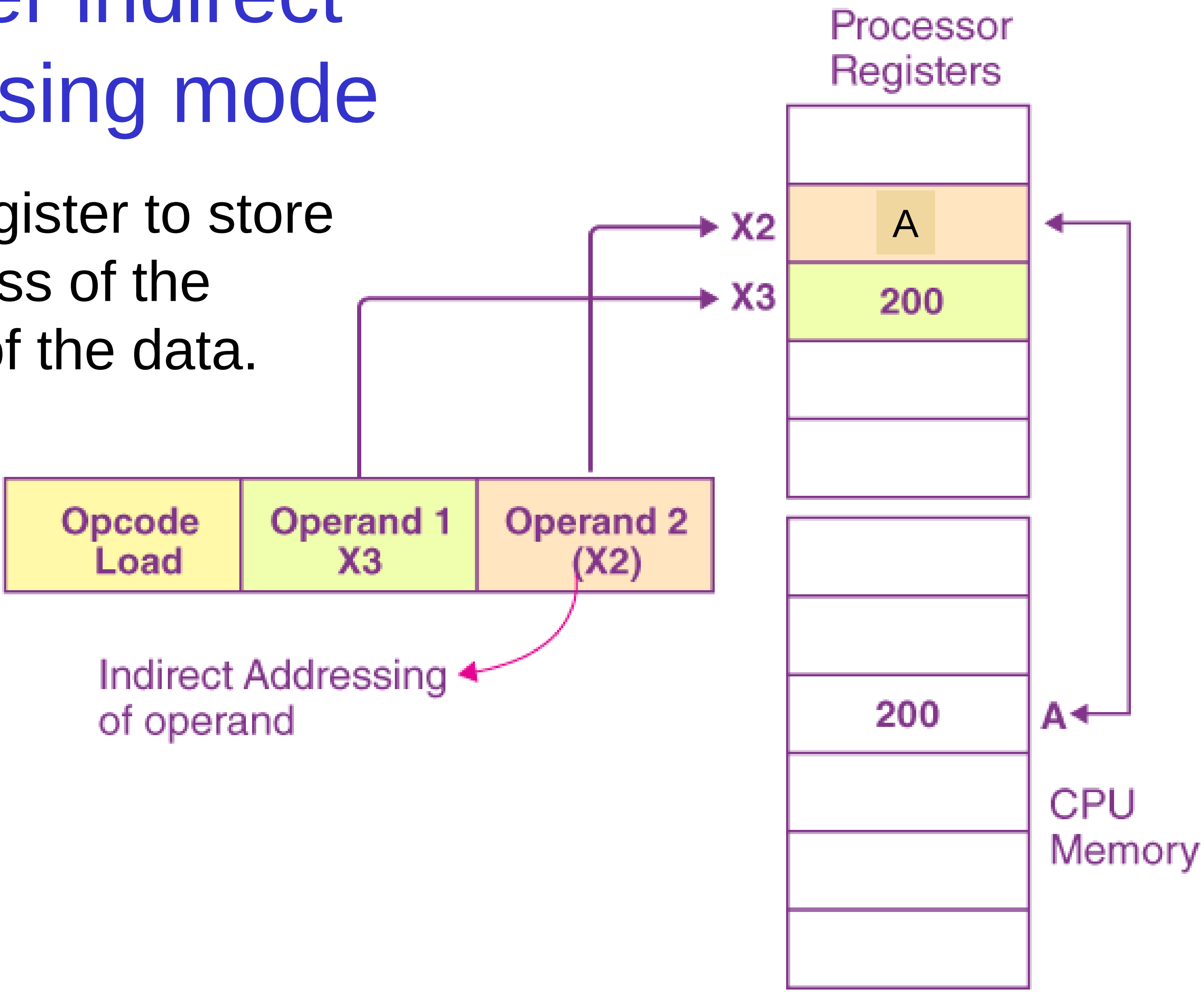
- *Register addressing* is where the data is located in a register.



- *Immediate addressing* is where the data is part of the instruction.
  - Add R2, #100

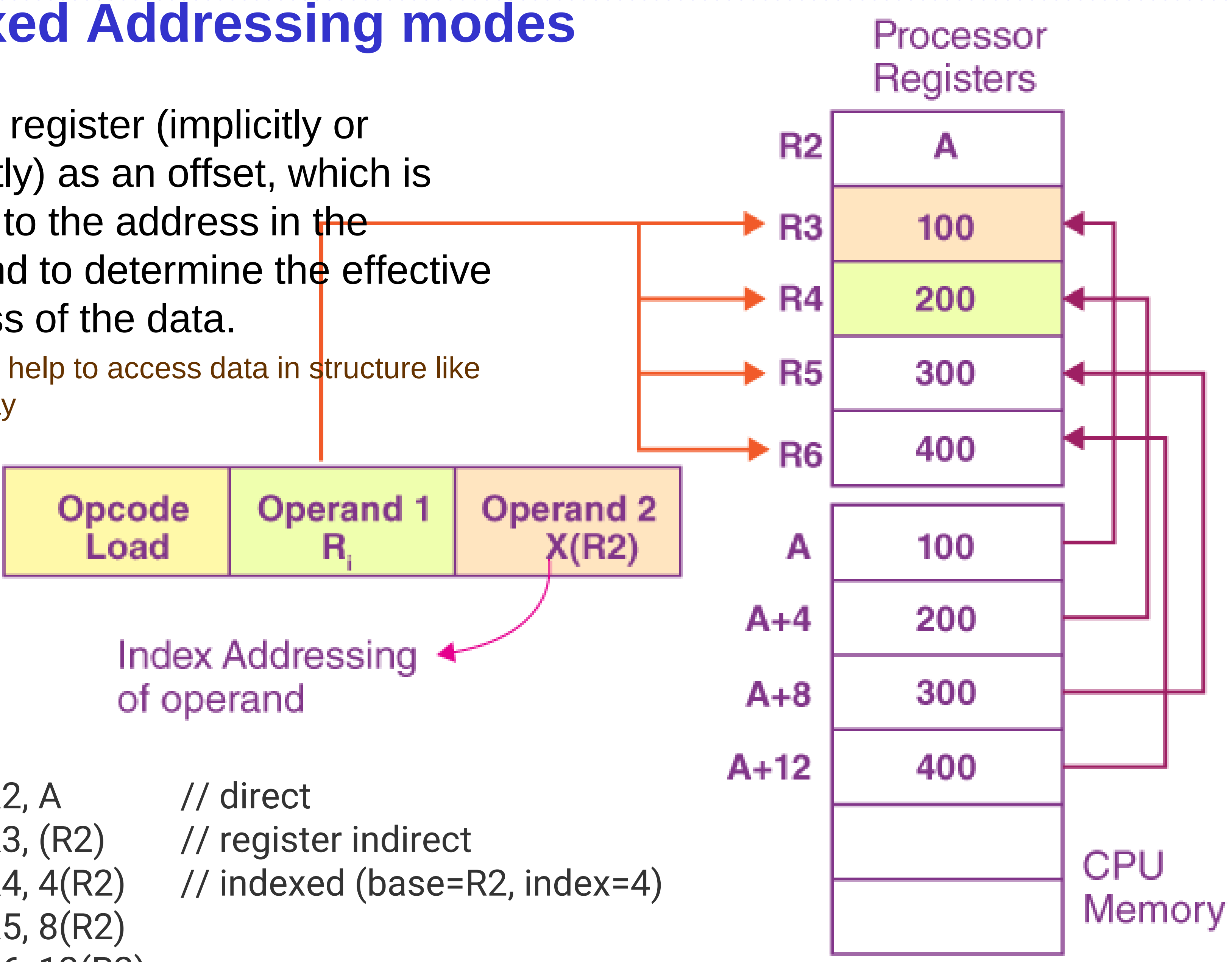
Register indirect addressing mode

- uses a register to store the address of the address of the data.
- Load X3, (X2)



Indexed Addressing modes

- uses a register (implicitly or explicitly) as an offset, which is added to the address in the operand to determine the effective address of the data.
  - Can help to access data in structure like Array



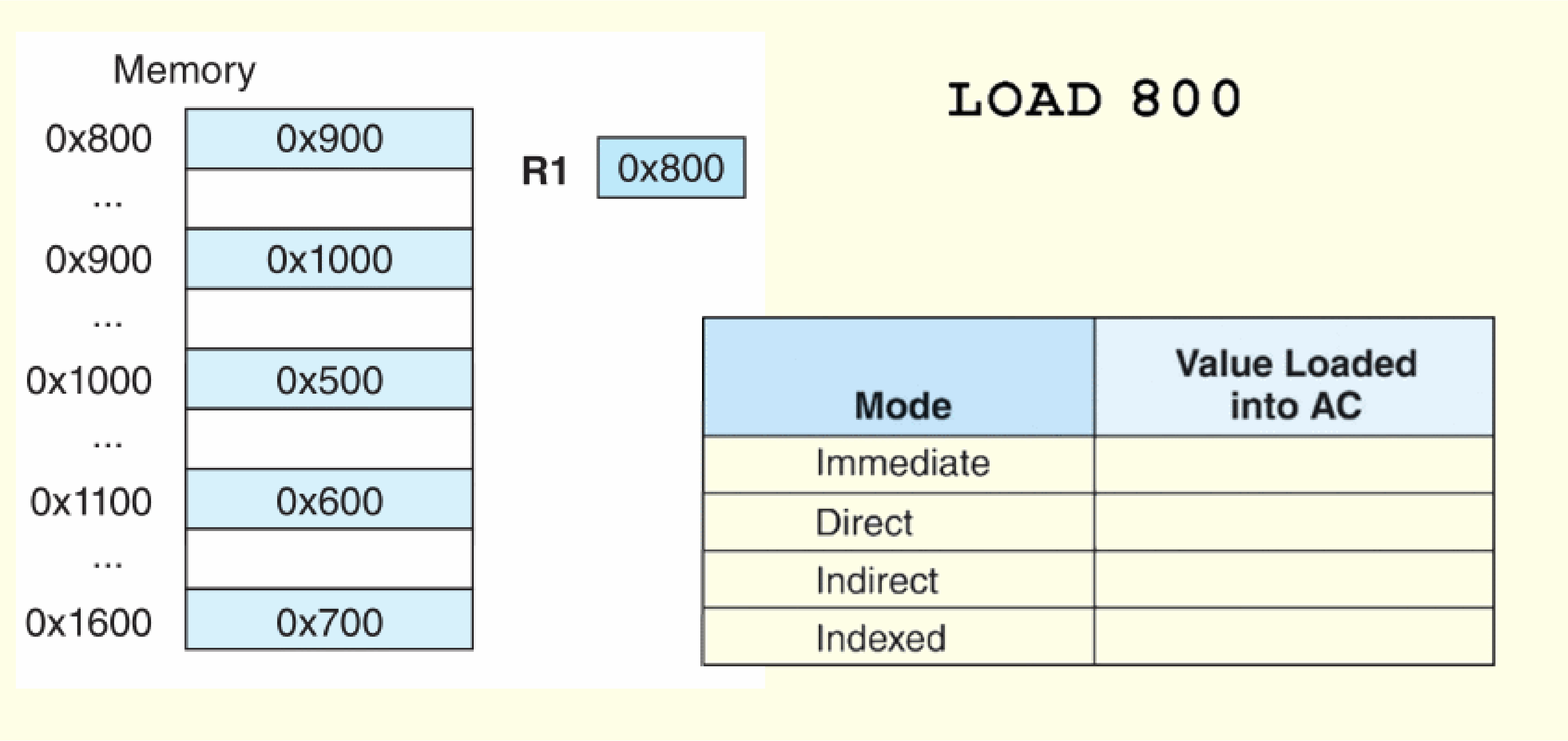
```
Load R2, A        // direct
Load R3, (R2)     // register indirect
Load R4, 4(R2)    // indexed (base=R2, index=4)
Load R5, 8(R2)
Load R6, 12(R2)
```

Other methods of addressing

- There are many variations to these addressing modes including:
    - Relative addressing
        - Similar to indexed which uses the operand as displacement from the base address (PC, base or index register)
    - Auto increment and decrement.
        - (R)+ , -(R) → Push vs Pop
        - Can be used to implement stack addressing
    - Indirect indexed addressing
- We won’t cover these in detail.

5.4 Addressing

- For the instruction shown, what value is loaded into the accumulator for each addressing mode?



Memory

0x800	0x900
...	
0x900	0x1000
...	
0x1000	0x500
...	
0x1100	0x600
...	
0x1600	0x700

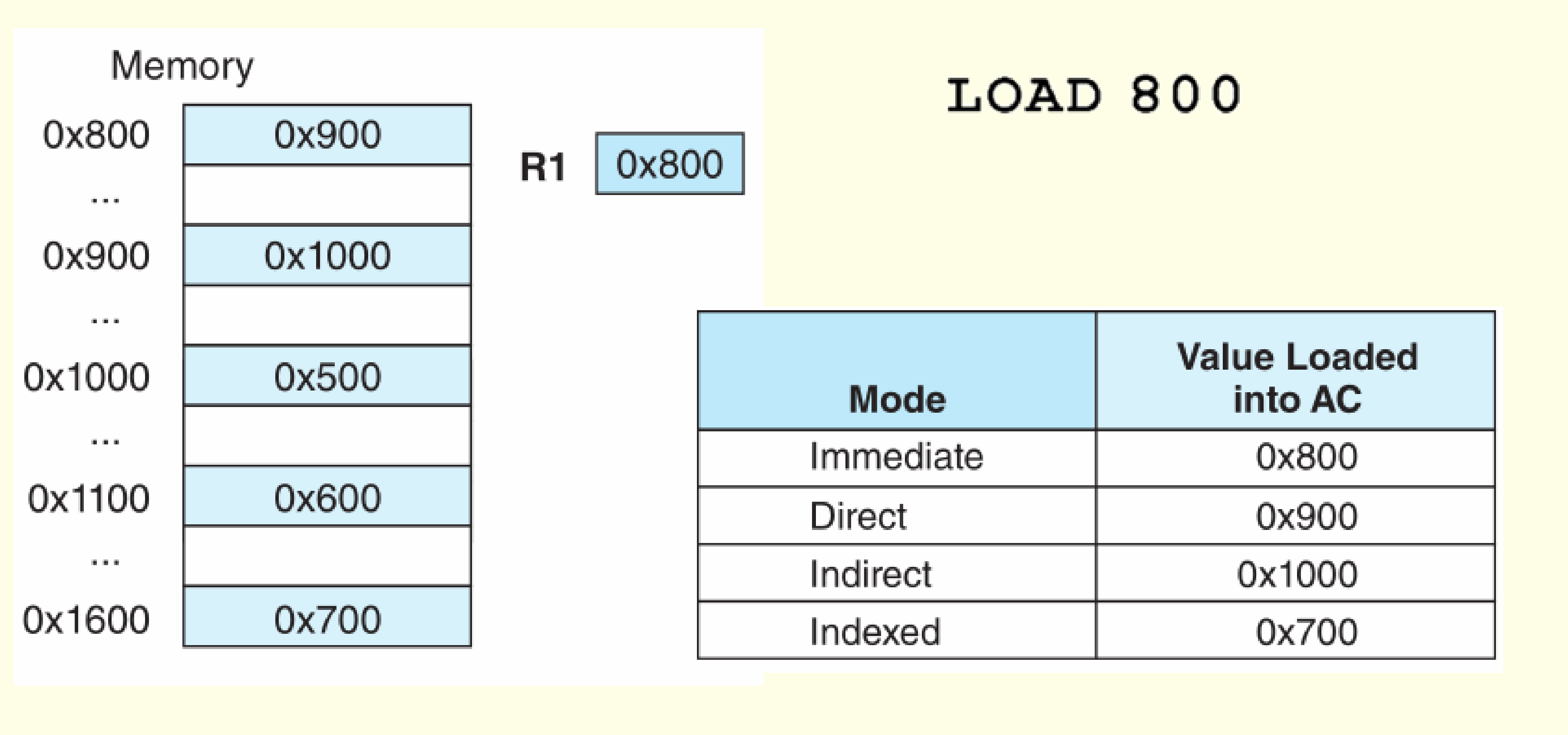
R1 0x800

```
LOAD 800
```

Mode	Value Loaded into AC
Immediate	
Direct	
Indirect	
Indexed	

Addressing

- For the instruction shown, what value is loaded into the accumulator for each addressing mode?



LOAD 800

Mode	Value Loaded into AC
Immediate	0x800
Direct	0x900
Indirect	0x1000
Indexed	0x700

Instruction-Level Pipelining (ILP)

http://www.intel.com/content/www/us/en/silicon-innovations/standards-14nm-explained-video.html

Instruction-Level Pipelining

- Some CPUs divide the fetch-decode-execute cycle into smaller steps.
- These smaller steps can often be executed in parallel to increase throughput.
- Such parallel execution is called *instruction-level pipelining* (*ILP*).

The next slide shows an example of instruction-level pipelining.

Example of executing small steps in parallel to increasing throughput: Laying the water pipe

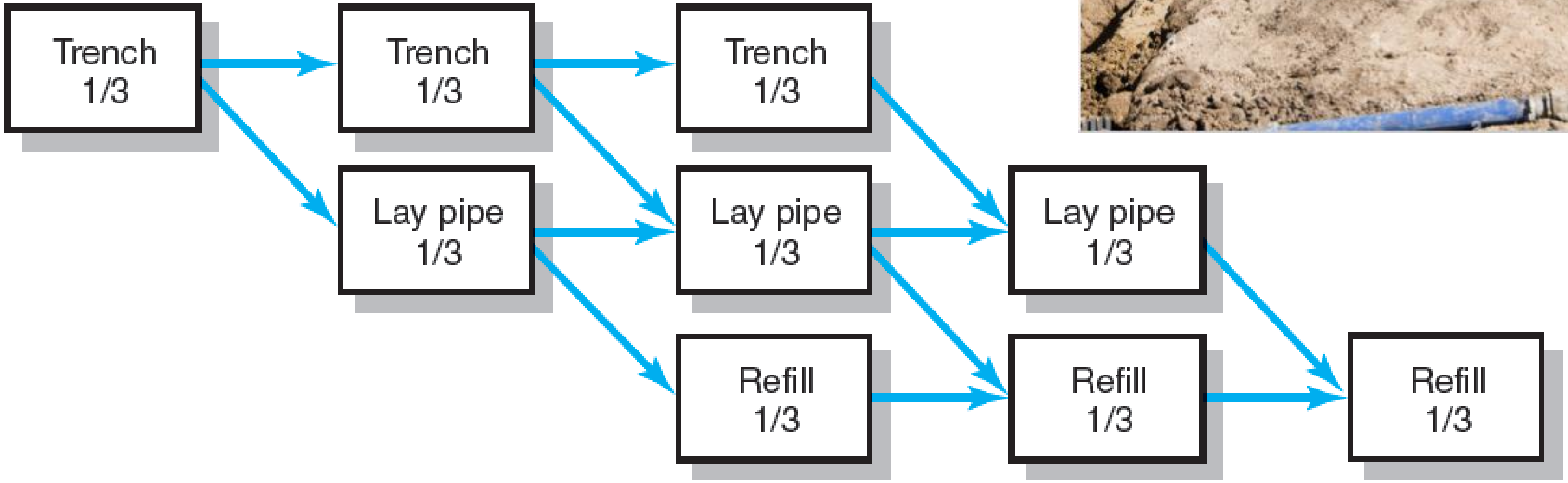
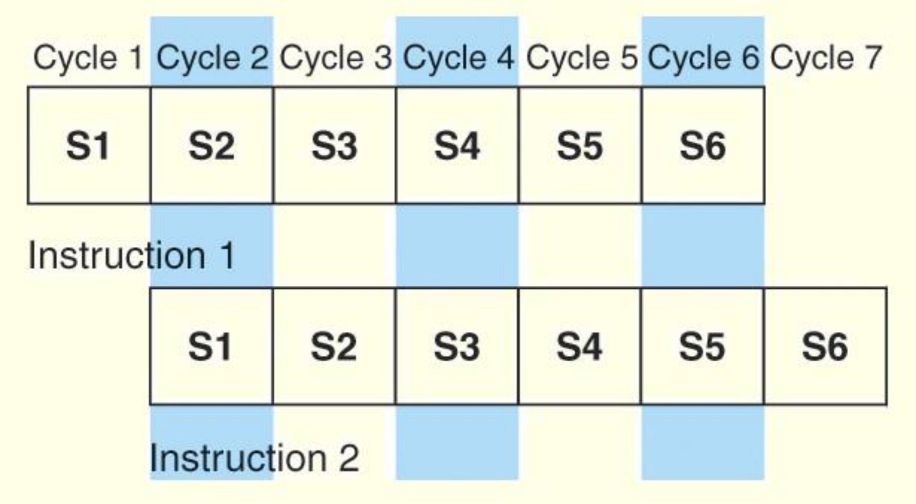


FIGURE 6.12

Instruction-Level Pipelining

- Suppose a fetch-decode-execute cycle were broken into the following smaller steps:

S1. Fetch instruction.
S2. Decode opcode.
S3. Calculate effective
address of operands.
S4. Fetch operands.
S5. Execute instruction.
S6. Store result.



	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7
Instruction 1	S1	S2	S3	S4	S5	S6	
Instruction 2		S1	S2	S3	S4	S5	S6

- So, we have *a six-stage pipeline*.
- For every clock cycle, one small step is carried out, and the stages are overlapped.

Instruction-Level Pipelining

- The theoretical speedup offered by a pipeline can be determined as follows, assume a program of ***n*** tasks:

  -Let t_p be the time per stage. Each instruction represents a task, T, in the pipeline.

  -The first task (instruction) requires $k \times t_p$ time to complete in *a k-stage pipeline*.

  -The remaining (n - 1) tasks emerge from the pipeline one per clock cycle. So the total time to complete the remaining tasks is $(n - 1)t_p$.

- Thus, to complete *n* tasks using a *k*-stage pipeline requires total time = $(k \times t_p) + (n - 1)t_p = (k + n - 1)t_p$.

1. For a 6-stage pipeline, how many stages do we need for 5 tasks?
2. How many clock cycle for the traditional Fetch-decode-execute?

Instruction-Level Pipelining

- If we take the time required to complete *n* tasks without a pipeline and divide it by the time it takes to complete *n* tasks using a pipeline, we find:

$$\text{Speedup } S = \frac{n t_n}{(k + n - 1)\, t_p}$$

- If we take the limit as *n* approaches infinity, (*k* + *n* - 1) approaches *n*, which results in a theoretical speedup of:

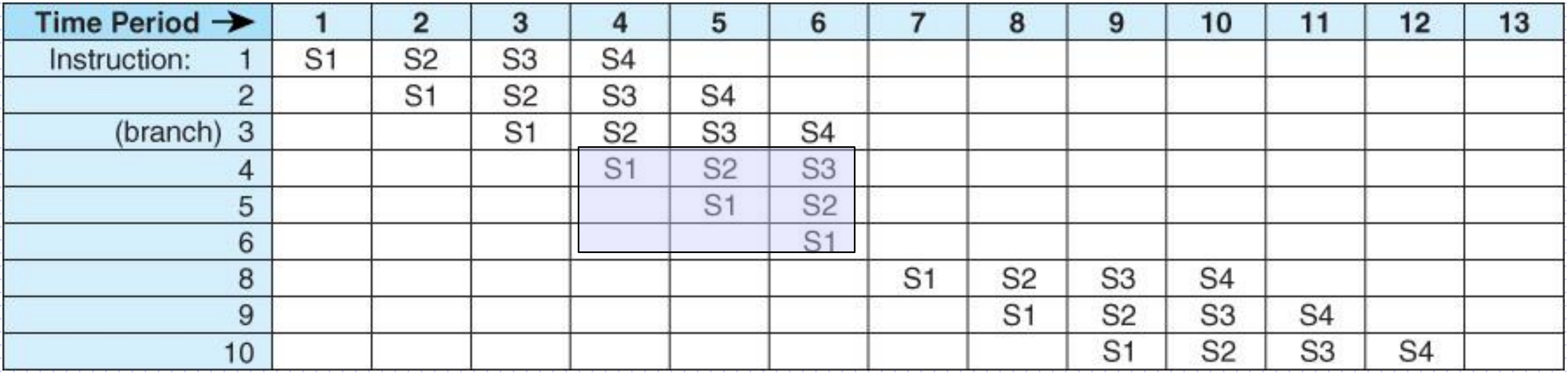
$$\text{Speedup } S = \frac{k t_p}{t_p} = k$$

Instruction-Level Pipelining

- Our neat equations take a number of things for granted.
  - First, we have to assume that the architecture supports fetching instructions and data in parallel.
  - Second, we assume that the pipeline can be kept filled at all times. This is not always the case.
- Pipeline *hazards* arise that cause pipeline conflicts and stalls.

Instruction-Level Pipelining

- An instruction pipeline may stall, or be flushed for any of the following reasons:
  - Resource conflicts– read versus write to memory.
  - Data dependencies– need result from other instruction.
  - Conditional branching
- Measures can be taken at the software level as well as at the hardware level to reduce the effects of these hazards, but they cannot be totally eliminated.



Time Period →	1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction: 1	S1	S2	S3	S4									
2		S1	S2	S3	S4								
(branch) 3			S1	S2	S3	S4							
4				S1	S2	S3							
5					S1	S2							
6						S1							
8							S1	S2	S3	S4			
9								S1	S2	S3	S4		
10									S1	S2	S3	S4	

Figure 05.05
Example Instruction Pipeline with Conditional Branch

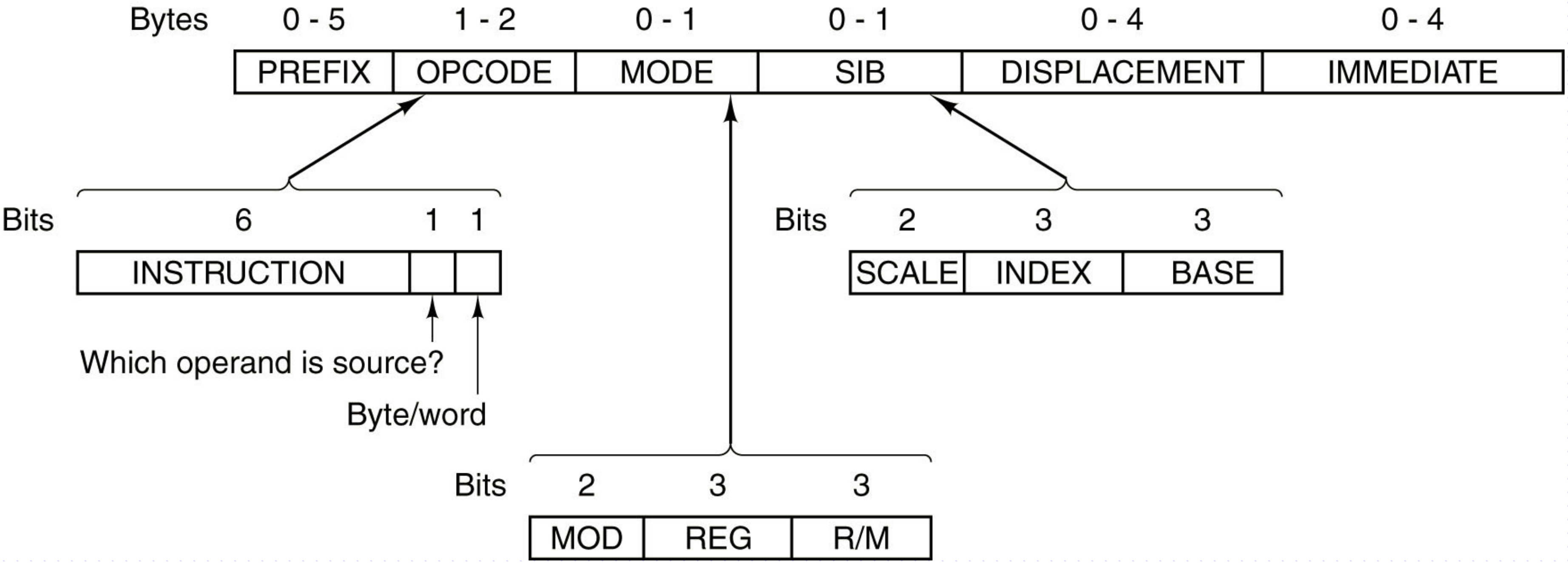
Real-World Examples of ISAs

- Intel processors support a wide array of addressing modes.
- The original 8086 provided 17 ways to address memory, most of them variants on the methods presented in this chapter.
- Owing to their need for backward compatibility, the Pentium chips also support these 17 addressing modes.
- The Itanium, having a RISC core, supports only one: register indirect addressing with optional post increment. – not successful.

Real-World Examples of ISAs

- Intel introduced pipelining to their processor line since the introduction of Pentium chip.
- The first Pentium (x86 32-bit architecture) had two five-stage pipelines. Each subsequent Pentium processor had a longer pipeline than its predecessor with the Pentium IV having a 24-stage pipeline.
- The Itanium (IA-64) has only a 10-stage pipeline.
- The Core i series has 5 - 20 stages pipeline.

The Pentium 4 Instruction Formats



- 17 addressing modes; Support 8, 16, 32 bits operand
- Flexible instruction format
- Compatible with the old 8080 CPU’s assembly

Real-World Examples of ISAs

- MIPS was an acronym for *Microprocessor Without Interlocked Pipeline Stages*.
- The architecture is little endian and word-addressable with three-address, fixed-length instructions.
  - Used in RISC architecture of Silicon Graphics (SGI)
    - Cisco routers, the original Sony PlayStations (R3000a),
    - PS5 uses AMD ZEN 2 (CISC x86 64 CISC architecture)
  - Still use in embedded system like printers (if not android)
- Mostly replaced by ARM
- Like Intel, the pipeline size of the MIPS processors has grown: The R2000 and R3000 have five-stage pipelines; the R4000 and R4400 have 8-stage pipelines.

Real-World Examples of ISAs

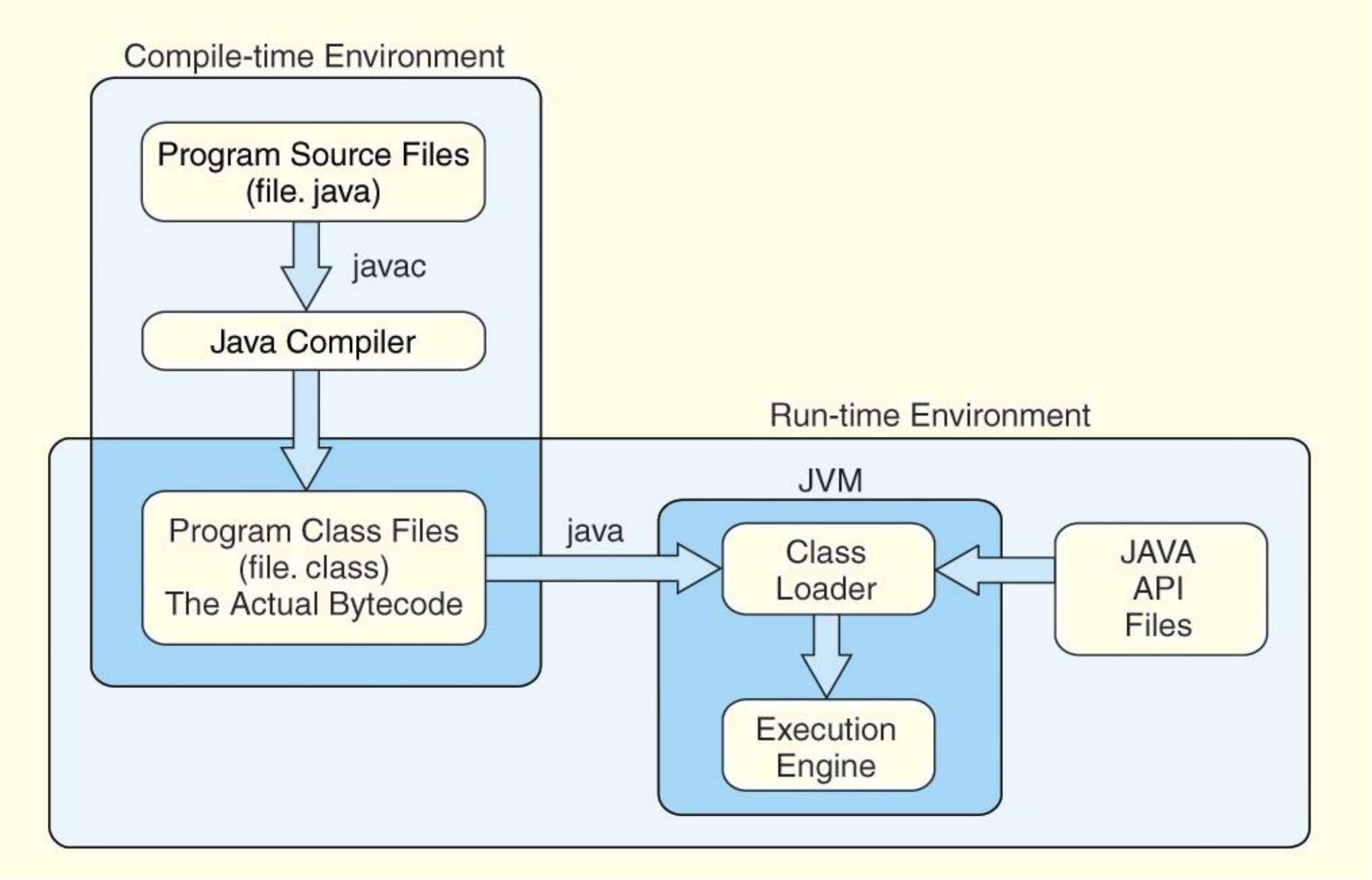
- The R10000 has three pipelines: A five-stage pipeline for integer instructions, a seven-stage pipeline for floating-point instructions, and a six-stage pipeline for `LOAD/STORE` instructions.
- In all MIPS ISAs, only the `LOAD` and `STORE` instructions can access memory.
- The ISA uses only base addressing mode.
- The assembler accommodates programmers who need to use immediate, register, direct, indirect register, base, or indexed addressing modes.

Real-World Examples of ISAs

- The Java programming language is an interpreted language that runs in a software machine called the *Java Virtual Machine* (JVM).
- A JVM is written in a native language for a wide array of processors, including MIPS, ARM and Intel.
- Like a real machine, the JVM has an ISA all of its own, called *bytecode*. This ISA was designed to be compatible with the architecture of any machine on which the JVM is running.

The next slide shows how the pieces fit together.

Real-World Examples of ISAs



Real-World Examples of ISAs

- Java bytecode is a stack-based language.
- Most instructions are zero address instructions.
- The JVM has four registers that provide access to five regions of main memory.
- All references to memory are offsets from these registers (indexed addressin). Java uses no pointers or absolute memory references.
- Java was designed for platform interoperability, not performance!

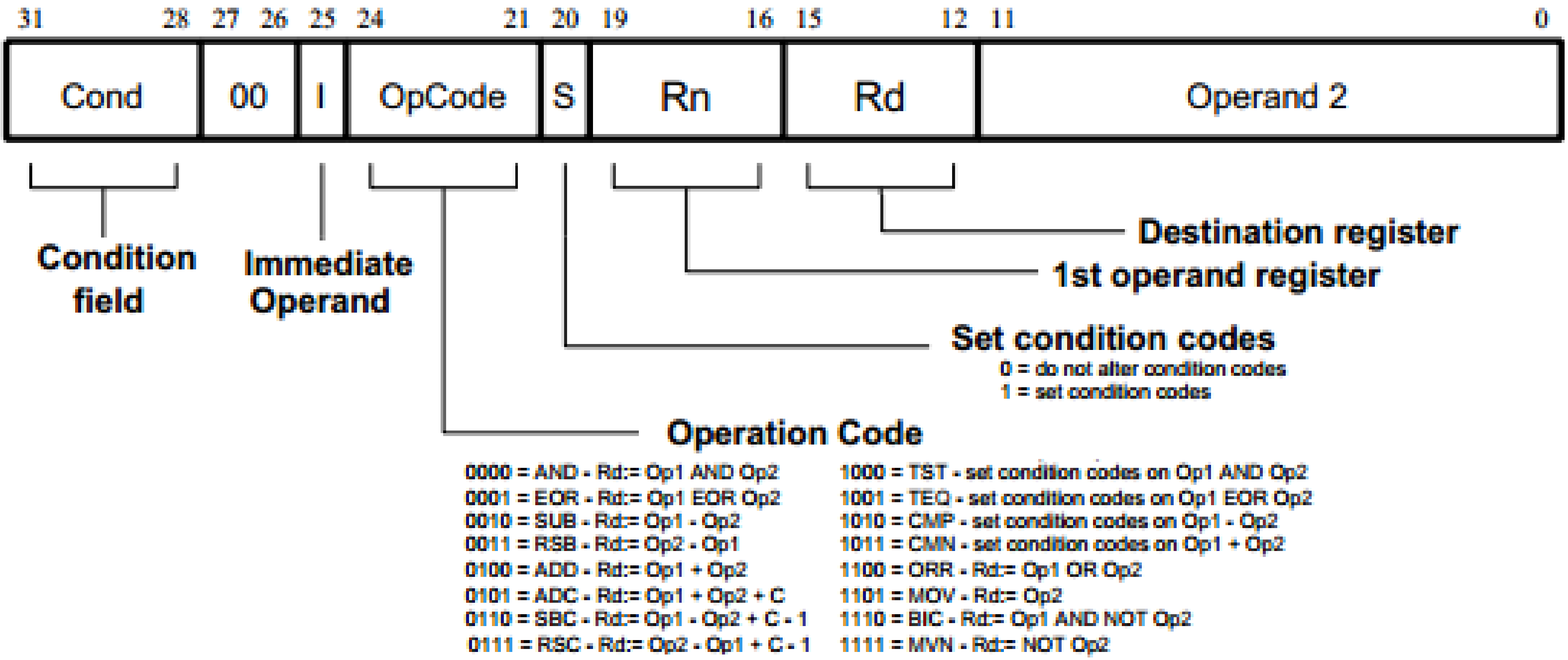
Real-World Examples of ISAs

- ARM is the most widely used 32-bit instruction architecture:
  - 95%+ of smartphones,
  - 80%+ of digital cameras
  - 40%+ of all digital television sets
- Founded in 1990, by Apple and others, ARM (Advanced RISC Machine) is now a British firm, ARM Holdings.
- ARM Holdings does not manufacture these processors; it sells licenses to manufacture.

Real-World Examples of ISAs

- ARM is a load/store architecture : all data processing must be performed on values in registers, not in memory.
- It uses fixed-length, three-operand instructions and simple addressing modes
- ARM processors have a minimum of a three-stage pipeline (consisting of fetch, decode, and execute);
  - Newer ARM processors have deeper pipelines (more stages). Some ARM8 implementations have 13-stage integer pipelines

ARM Instruction format for the 32-bit version.



31 28	27 26	25	24 21	20	19 16	15 12	11 0
Cond	00	I	OpCode	S	Rn	Rd	Operand 2

0 = do not alter condition codes
1 = set condition codes

Operation Code

0000 = AND - Rd:= Op1 AND Op2	1000 = TST - set condition codes on Op1 AND Op2
0001 = EOR - Rd:= Op1 EOR Op2	1001 = TEQ - set condition codes on Op1 EOR Op2
0010 = SUB - Rd:= Op1 - Op2	1010 = CMP - set condition codes on Op1 - Op2
0011 = RSB - Rd:= Op2 - Op1	1011 = CMN - set condition codes on Op1 + Op2
0100 = ADD - Rd:= Op1 + Op2	1100 = ORR - Rd:= Op1 OR Op2
0101 = ADC - Rd:= Op1 + Op2 + C	1101 = MOV - Rd:= Op2
0110 = SBC - Rd:= Op1 - Op2 + C - 1	1110 = BIC - Rd:= Op1 AND NOT Op2
0111 = RSC - Rd:= Op2 - Op1 + C - 1	1111 = MVN - Rd:= NOT Op2

- Fixed length, 3-operand ISA

Real-World Examples of ISAs

- ARM has 37 total registers but their visibility depends on the processor mode.
  - Intel Pentium, there were up to 128 registers
- ARM allows multiple register transfers.
  - It can simultaneously load or store any subset of the16 general-purpose registers from/to sequential memory addresses.
- Control flow instructions include unconditional and conditional branching and procedure calls
- Most ARM instructions execute in a single cycle, provided there are no pipeline hazards or memory accesses.

Conclusion

- Instruction format can be flexible and complex (CISC) or simple format (RISC)
- Instruction could be equipped with many possible addressing mode for fetching operands.
- Instruction can be speeded up using instruction-level pipelining.
- In real-world, various approach of ISA have been evolved, Intel, AMD, MIPS, ARM, even the virtual machine like JAVA JVM.

Further reading

- Addressing mode https://byjus.com/gate/index-addressing-mode-notes/
- ISA definition https://www.youtube.com/watch?v=6fgbLOL7bis