

x86 Architecture & Its Assembly language programming

Dr A Sahu
Dept of Computer Science &
Engineering
IIT Guwahati

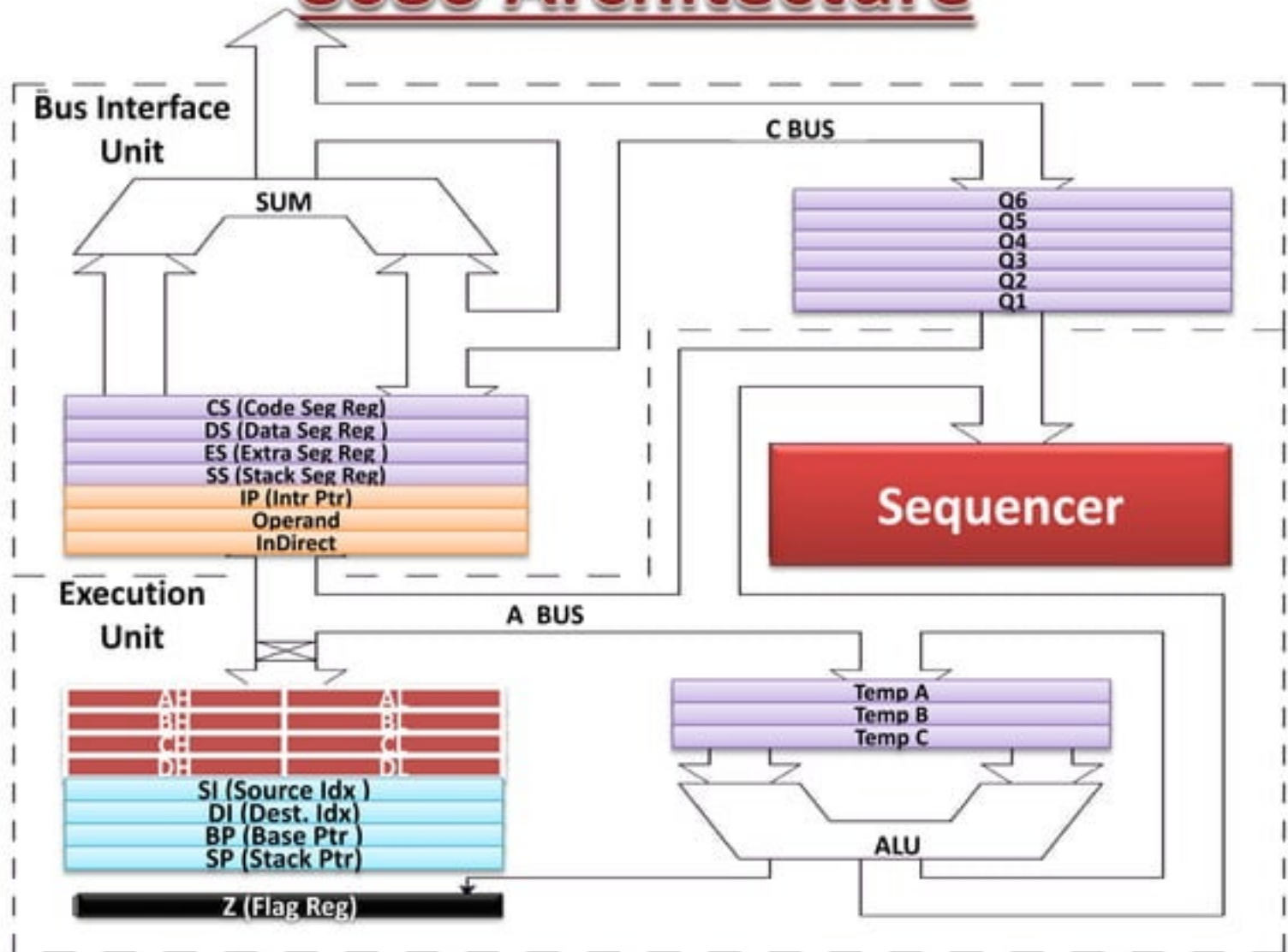
Outline

- Review of 8086 Architecture
 - Block diagram (Data Path)
- **Similarity with x86 (i386, Pentium, etc)**
 - **Very IMP for interview/knowledge**
 - **Not part of Examination**
- x86 Assembly language program
 - Memory model
 - Example programs
 - Data Segment
 - Loop and Nested Loop
- Next Class: Detail of assembly language
 - Summary of 8085/8086/i386 Arch & programming

Introduction to 8086 & i386 processor

- 16 bit Microprocessor
- All internal registers as well as internal and external data buses were 16 bits wide
- 4 Main Register, 4 Index Register, 4 Segment Register, Status Reg, Instr Ptr.
- Not compatible with 8085, but with successors
- Two Unit works in parallel:
 - Bus Interface Unit (BIU)
 - Execution Unit (EI)

8086 Architecture



8086 Architecture

- Execution Unit :
 - ALU may be loaded from three temp registers (TMPA, TMPB, TMPC)
 - Execute operations on bytes or 16-bit words.
 - The result stored into temp reg or registers connected to the internal data bus.
- Bus Interface Unit
 - BIU is intended to compute the addresses.
 - Two temporary registers
 - indirect addressing
 - four segment registers (DS, CS, SS and ES),
 - Program counter (IP - Instruction Pointer),
 - A 6-byte Queue Buffer to store the pre-fetched opcodes and data.
 - This Prefetch Queue optimize the bus usage.
 - To execute a jump instruction the queue has to be flushed since the pre-fetched instructions do not have to be executed.

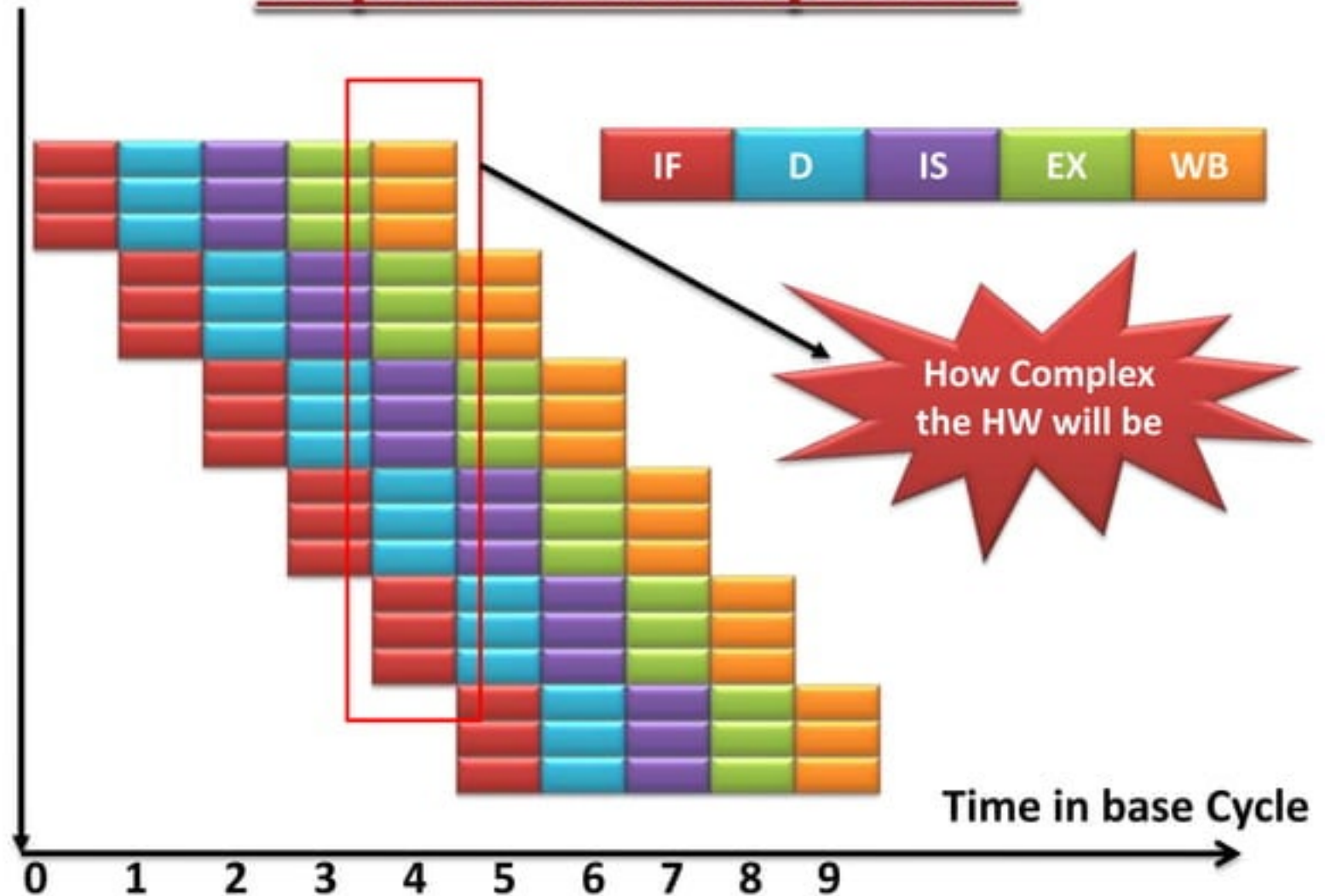
History of Intel Architectures

- **1978: 8086 (16 bit architecture)**
- **1980: 8087**
 - Floating point coprocessor is added
- **1982: 80286**
 - Increases address space to 24 bits
- **1985: 80386:**
 - 32 bits Add,
 - Virtual Mem & new add modes
 - Protected mode (OS support)
- **1989-95: 80486/Pentium/Pro**
 - Added a few instructions of base MMX

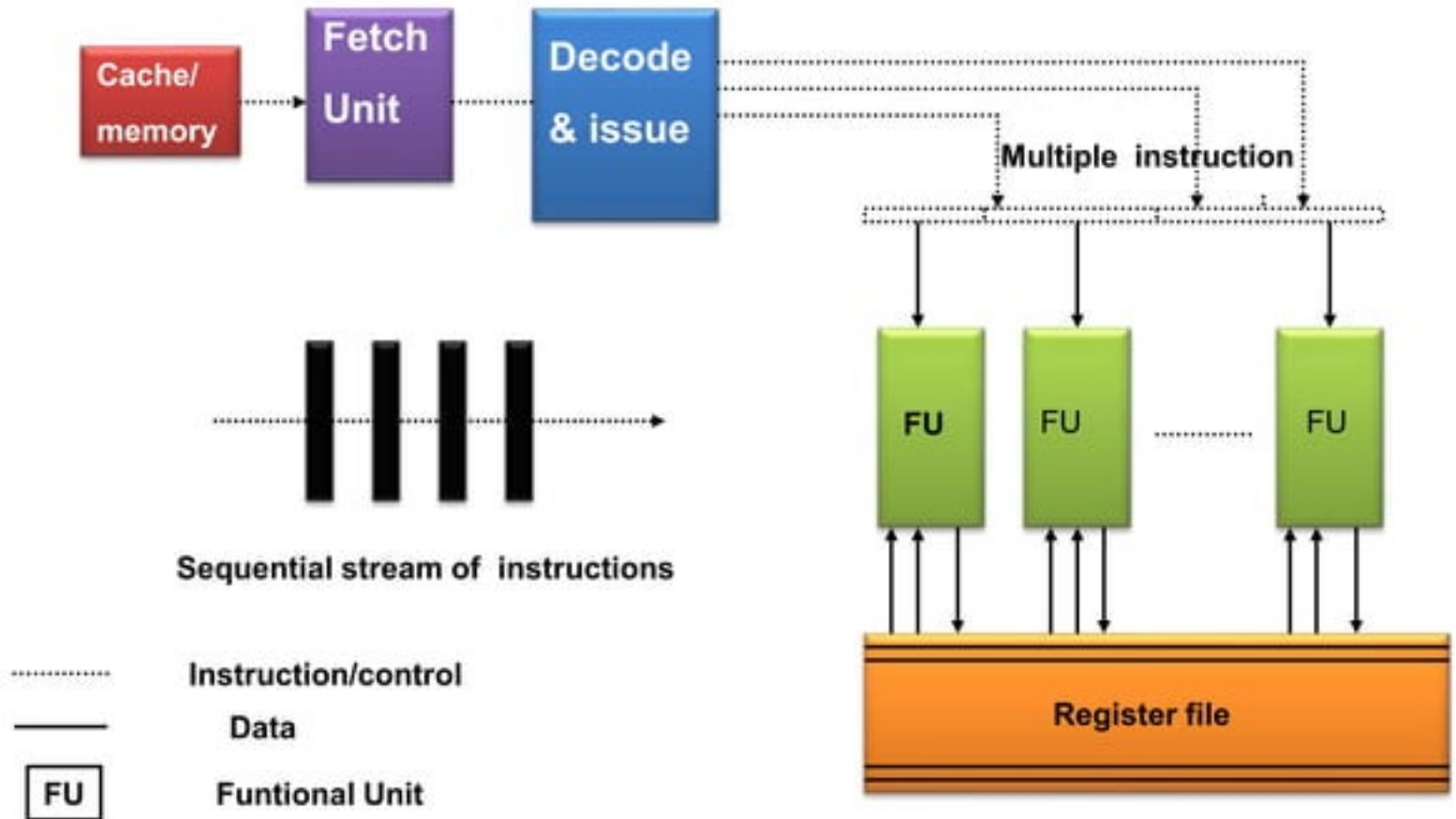
History of Intel Architectures

- **1997: Pentium II**
 - 57 new “MMX” instructions are added,
- **1999: Pentium III:**
 - Out of Order, added another 70 Streaming SIMD Ext (SSE)
- **2001: Pentium 4**
 - Net burst, another 144 instructions (SSE2)
- **2003: PI4 HT, Trace Cache**
- **2005: Centrino, low power**
- **2007: Core architecture, Duo**
- **2008: Atom, Quad core with HT....**
- **2009---:Multi core (Large chip multiprocessor)**

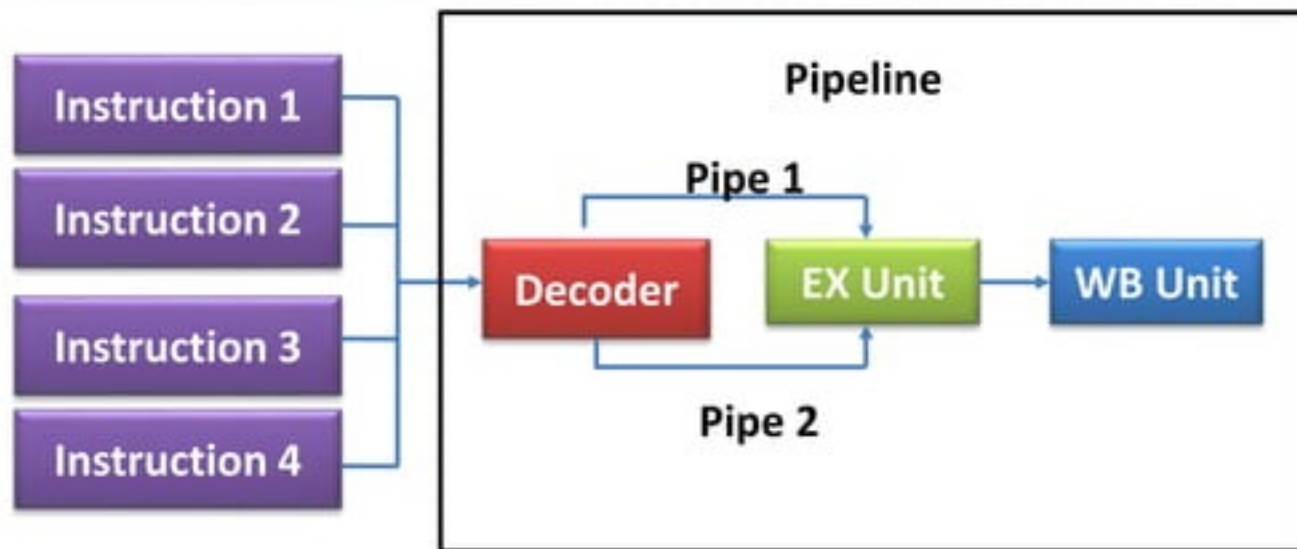
Superscalar Pipeline



ILP in Superscalar processors

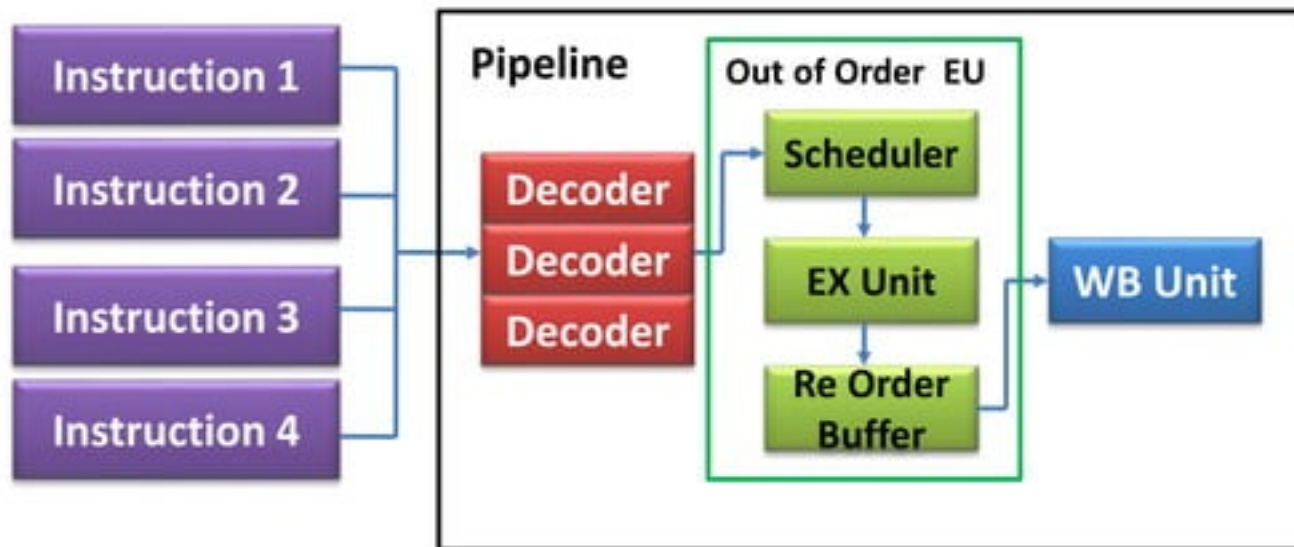


Intel P5 Architecture (Generation 5)



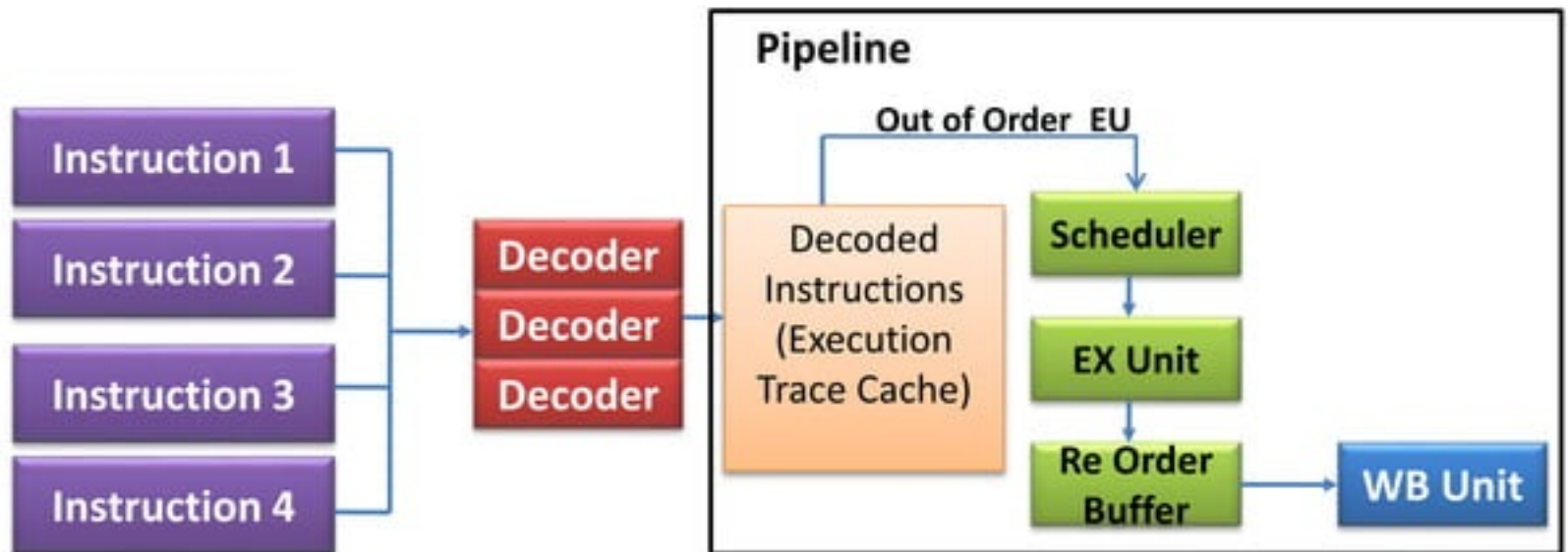
- Used in initial Pentium processor
- Could execute up to 2 instructions simultaneously
- Instructions sent through the pipeline in order - if the next two instructions had a dependency issue, only one instruction (pipe) would be executed and the second execution unit (pipe) went unused for that clock cycle.

Intel P6 Architecture (Generation 6)



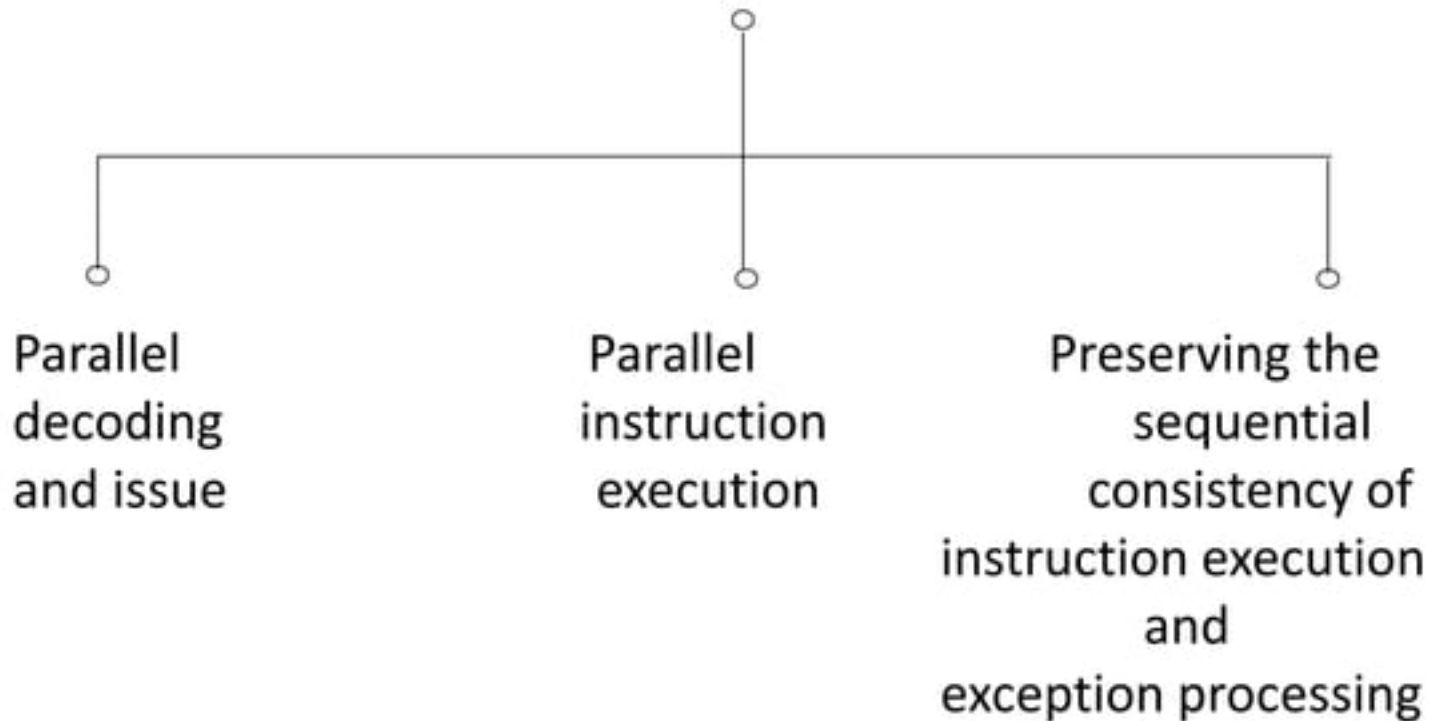
- Used in the Pentium II, III and Pro processors
- 3 instruction decoders, which break each CISC instruction (macro-op) into equivalent micro-operations (μ ops) for the Out-of-Order Execution unit
- 10 stage instruction pipeline utilized in this architecture

Intel NetBurst MicroArchitecture

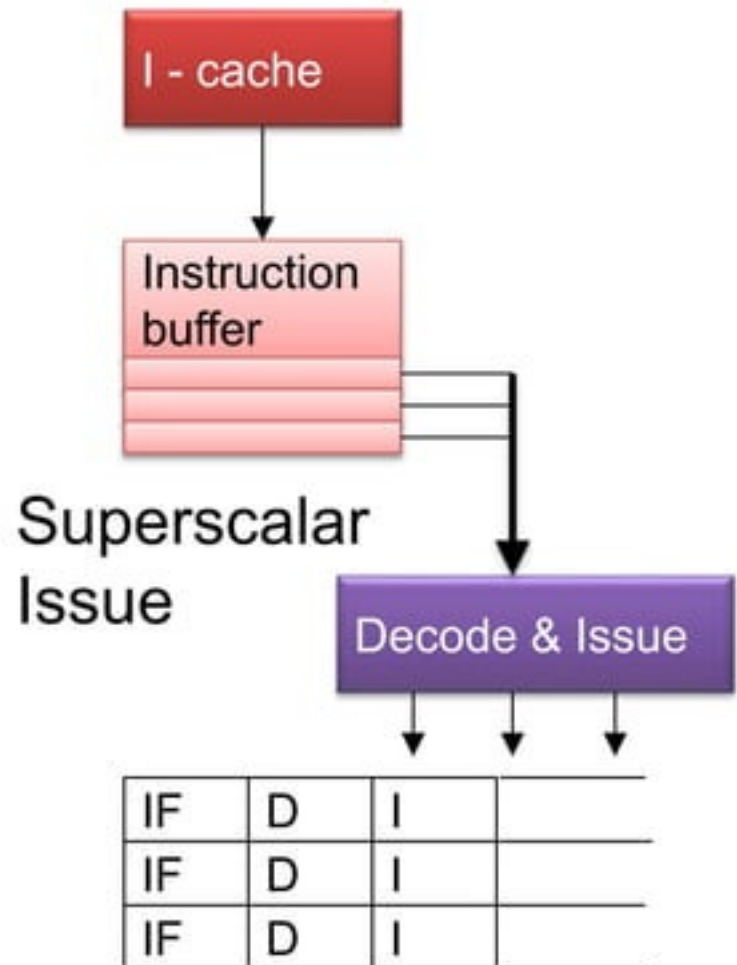
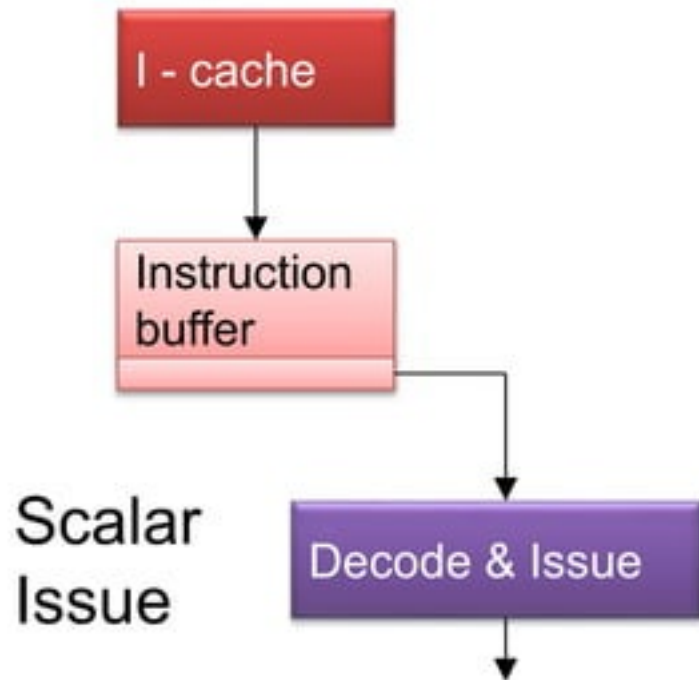


- New architecture used for the Intel Pentium IV and Pentium Xeon processors

Tasks of superscalar processing








Superscalar decode and issue



Parallel Decoding

- Fetch multiple instructions in instruction buffer
- Decode multiple instructions in parallel – instruction window
- Possibly check dependencies among these as well as with the instructions already under execution

Dependent/Independent Instructions

- ADD T A B $T = A + B$ 
 - ADD W C D $W = C + D$ 
 - LD A, 0(W) $A = M[W]$ 
 - ST C, 0(B) $M[B] = C$ 
- 

Read After Write (RAW), W after W, W after R

RAW (Ins2-Ins3): True dependency

WAW, WAR (Ins1 to Ins3) : false dependency

Issue vs Dispatch

Blocking Issue

- Decode and issue to EU

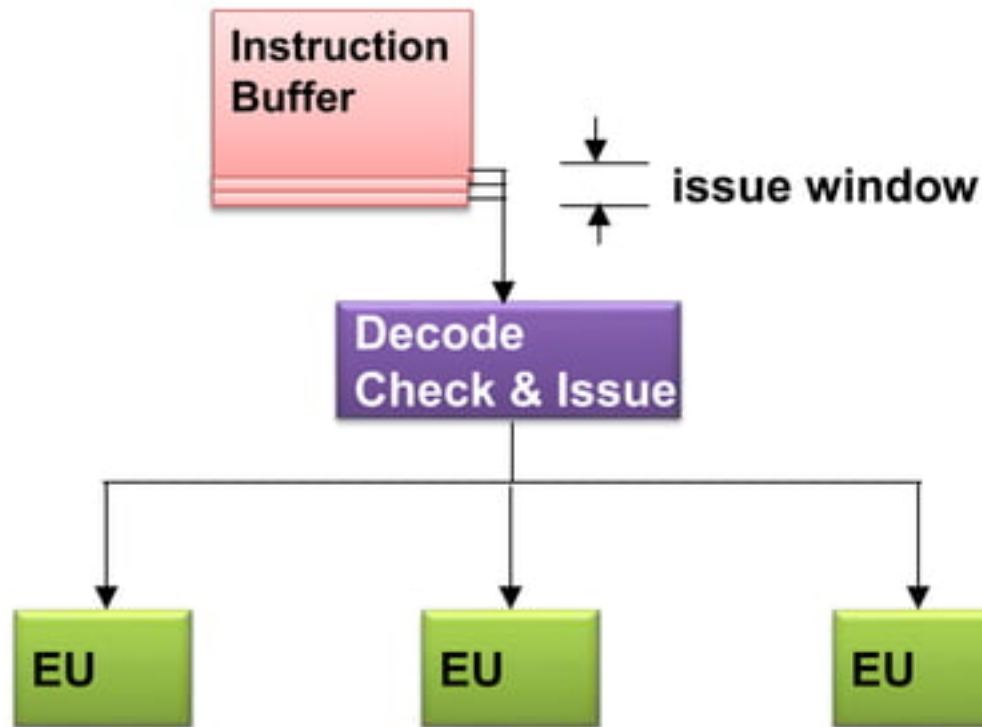
Instructions may be
blocked due to data
dependency

Non-blocking Issue

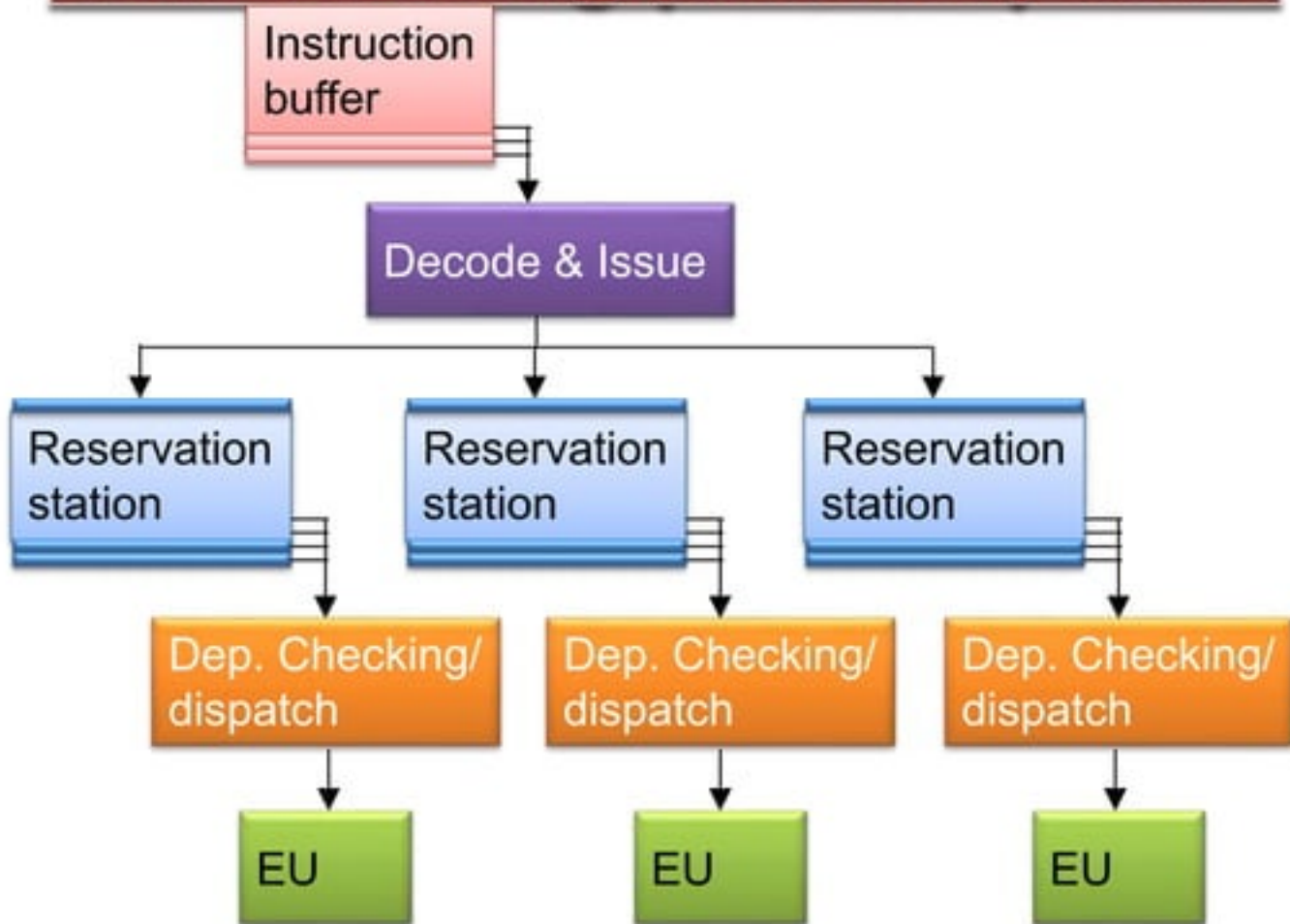
- Decode and issue to buffer
- From buffer dispatch to EU

Instructions are not
blocked due to data
dependency

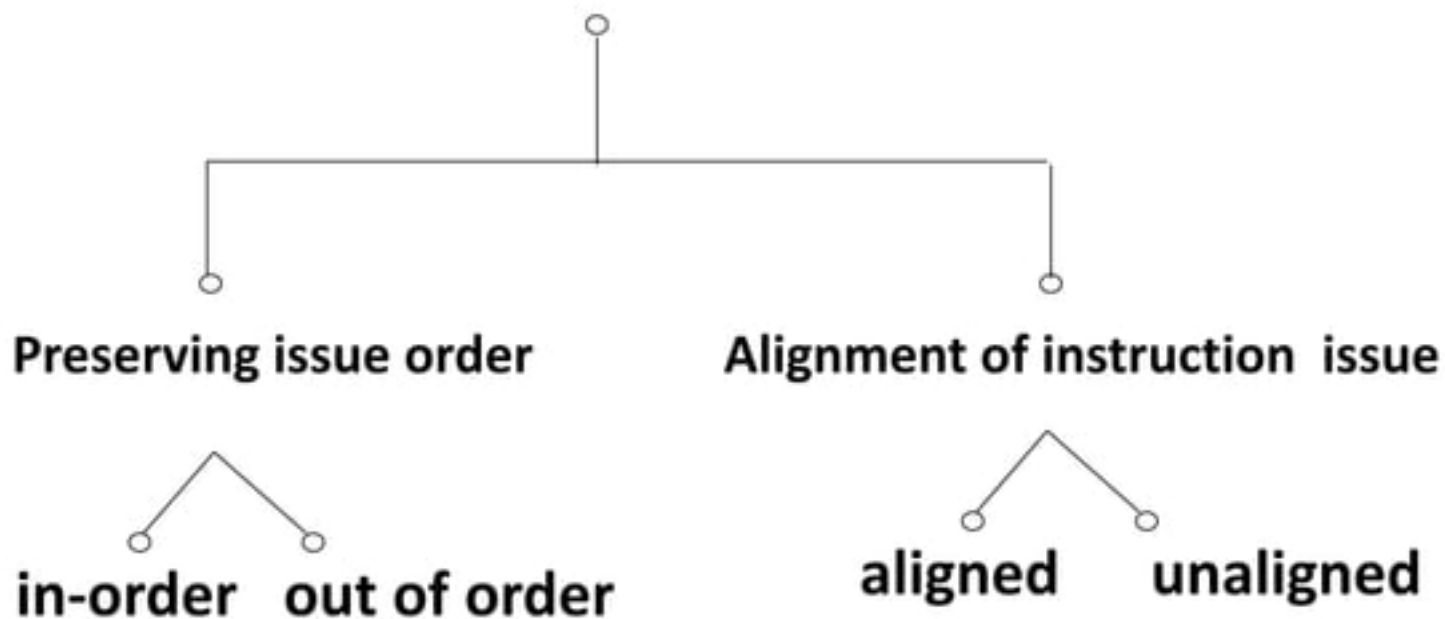
Blocking Issue








Non-blocking (shelved) Issue



Handling of Issue Blockages

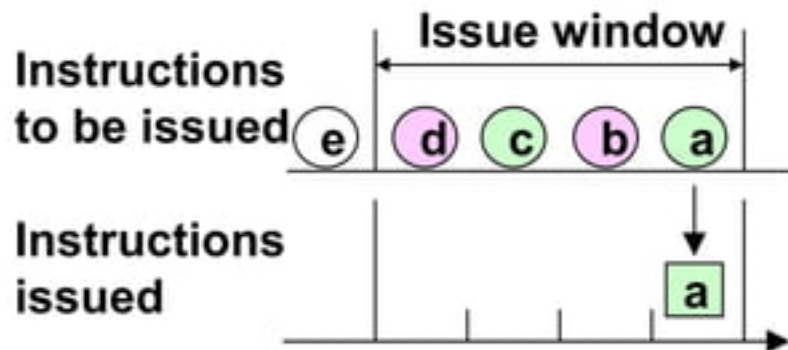


Dependent/Independent Instructions

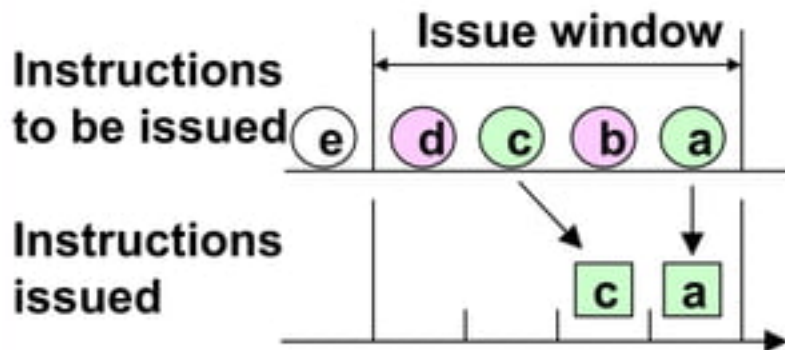
- ADD T A B $T = A + B$ 
 - ADD W C D $W = C + D$ 
 - LD A, 0(W) $A = M[W]$ 
 - ST C, 0(B) $M[B] = C$ 
- 

Issue Order




Issue in strict program order



Out of order Issue



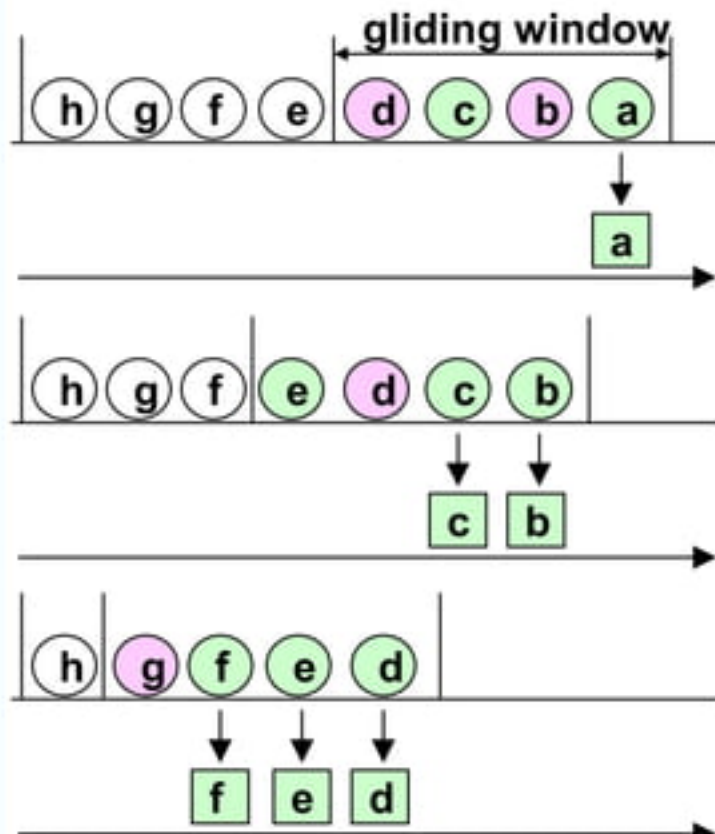
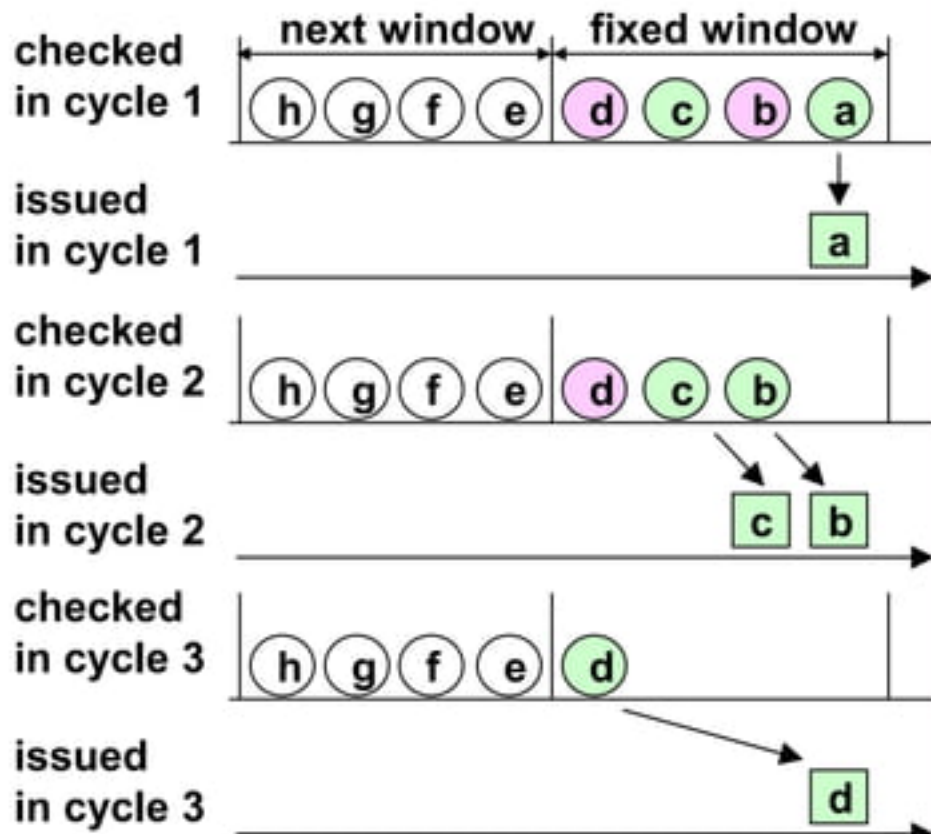
Example: MC 88110, PowerPC 601

-  Independent instruction
-  Dependent instruction
-  Issued instruction

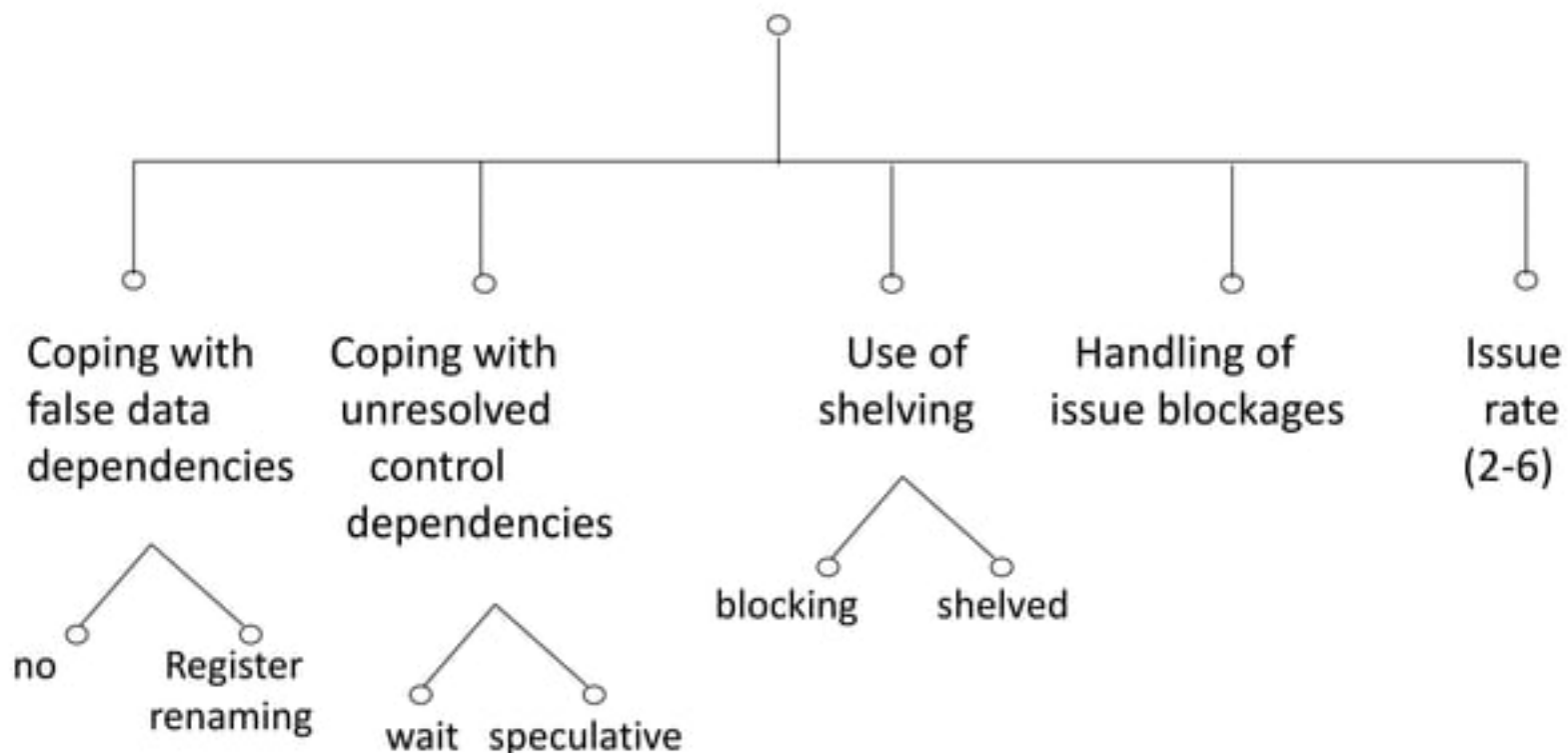
Alignment

Aligned Issue

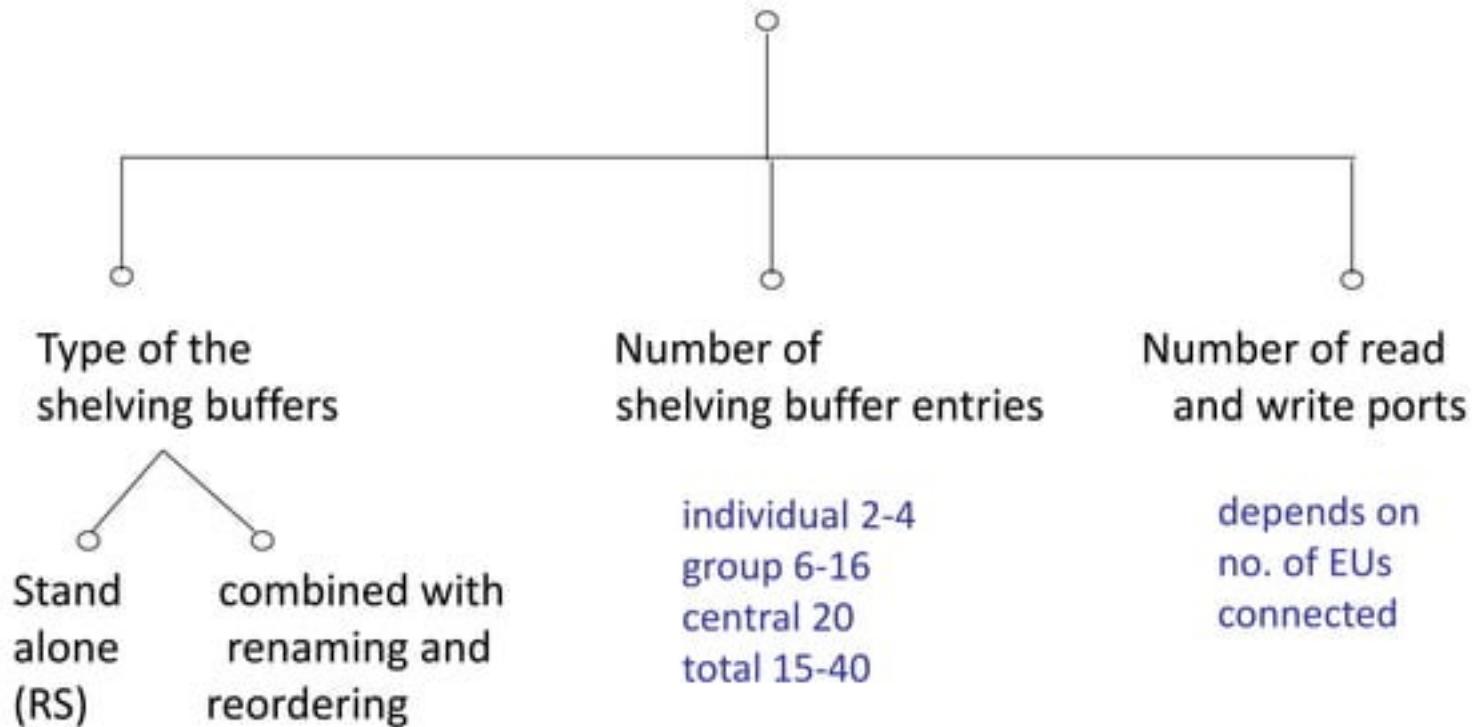
Unaligned Issue



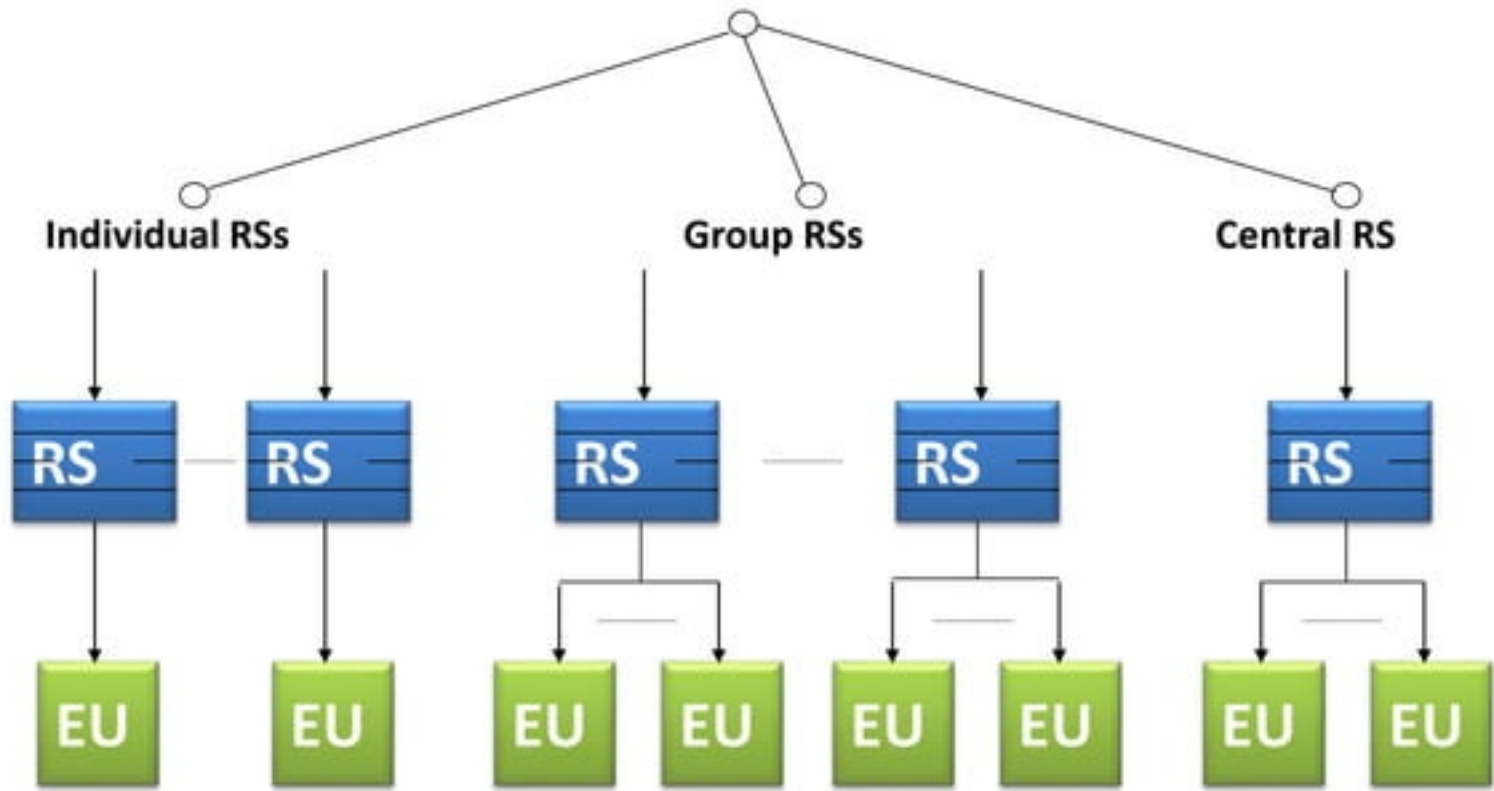
Design choices in instruction issue



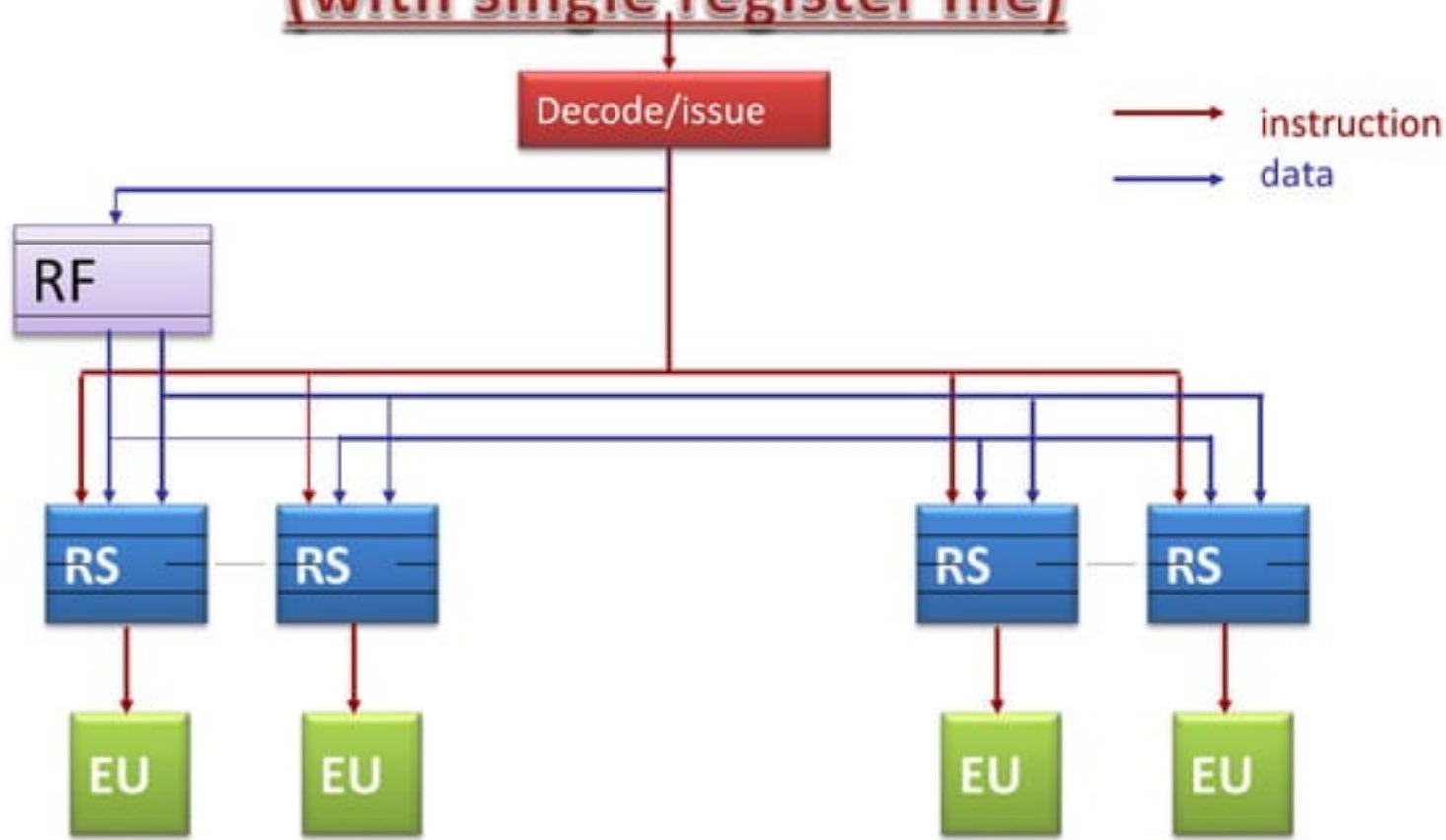
Layout of Shelving Buffers



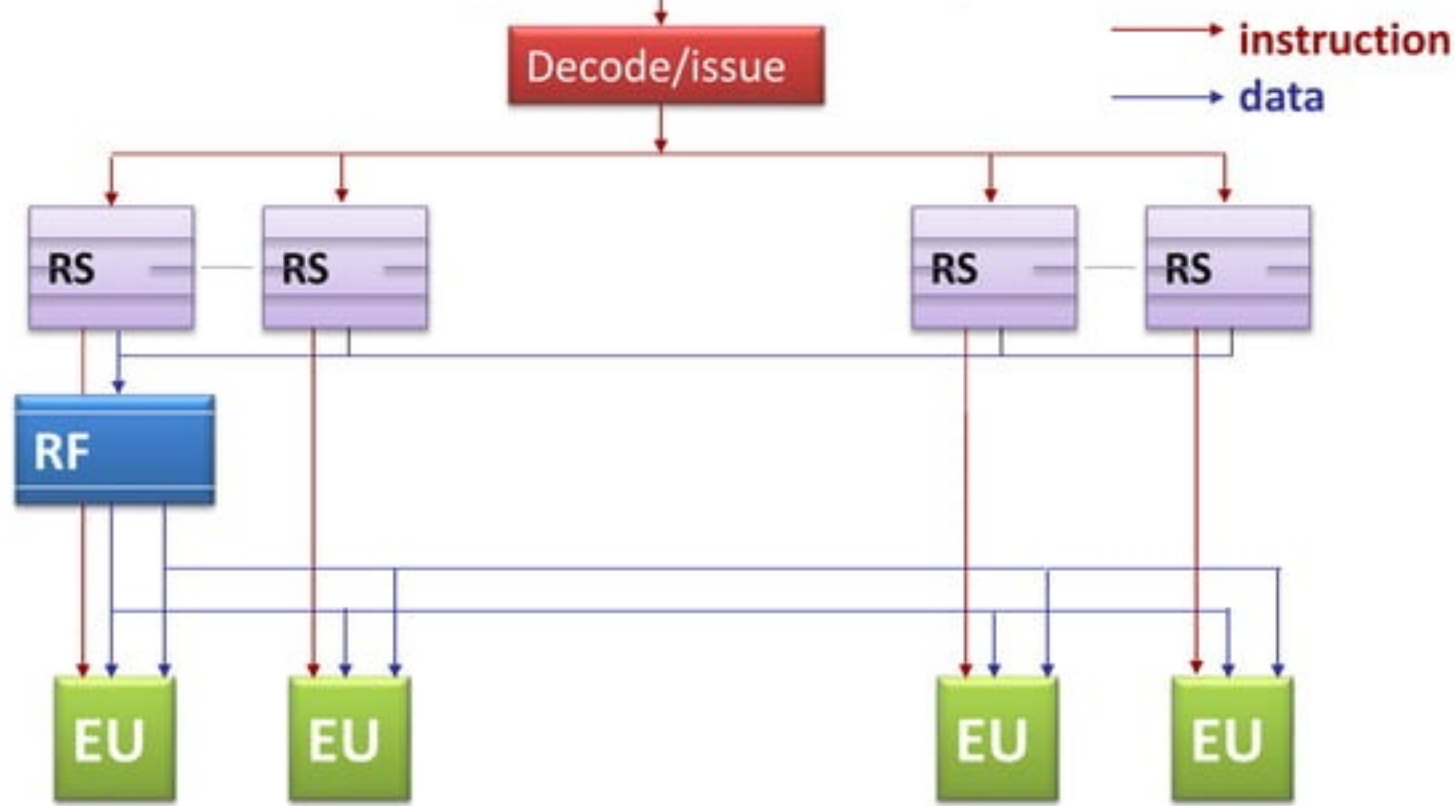
Reservation Stations (RS)



Issue bound operand fetch (with single register file)



Dispatch bound operand fetch (with single register file)



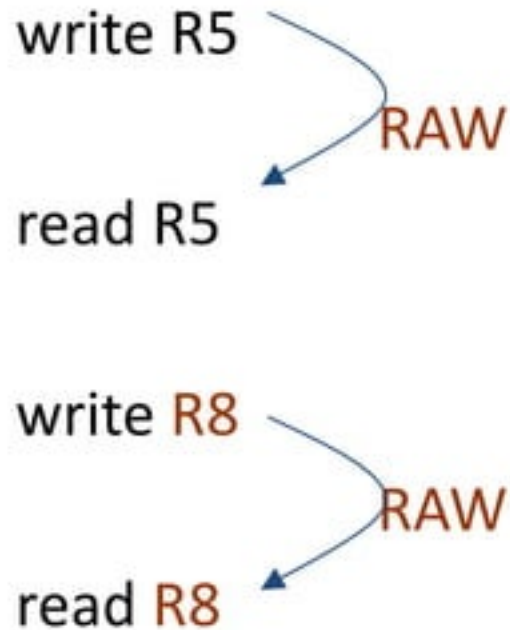
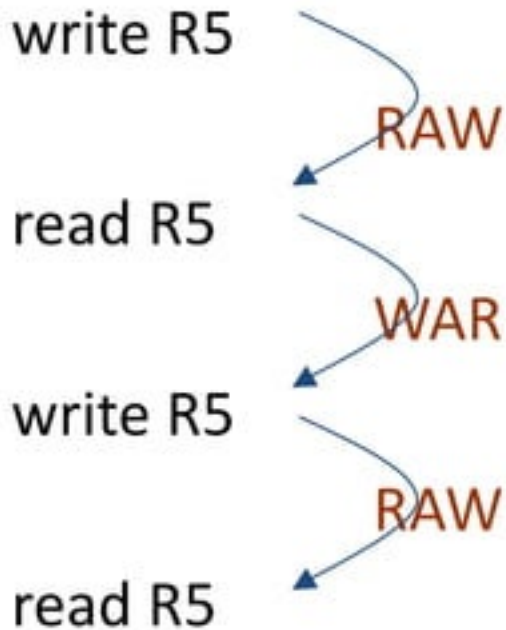
Why Renaming and Reordering?

- Register Renaming
 - Removes false dependencies (WAR and WAW)
- Reordering Buffer (ROB) : ***Pentium Out of order instruction processing***
 - Ensures sequential consistency of interrupts (precise vs imprecise interrupts)
 - Facilitates speculative execution
 - Branch execution
 - Execute both path and discard after getting CC Value

RAW, WAR and WAW (in Superscalar)



Register renaming



Who does renaming?

- Compiler
 - Done statically
 - Limited by registers visible to compiler
- Hardware
 - Done dynamically
 - Limited by registers available to hardware

X86 Assembly language program

- Kernel code writing
 - Process switching code
 - Thread synchronization code
 - Lock, barrier (test & set, fetch & increment, xcng)
 - **pthread spin_lock (in ASM) is 28% faster than intel tbb::spin_mutex (in C)**
- Time critical coding (Coding for DSP)
 - /usr/include/asm-i386
- Use of MASM/TASM/GCC/NASM compiler
 - gcc -S test.c -o test.s
- C/C++ code with asm block
- 8086 compatible with ii386/pentium

8086 Registers

- **AX** - the accumulator register (divided into **AH** / **AL**)
- **BX** - the base address register (divided into **BH** / **BL**)
- **CX** - the count register (divided into **CH** / **CL**)
- **DX** - the data register (divided into **DH** / **DL**)

- **SI** - source index register.
- **DI** - destination index register.
- **BP** - base pointer.
- **SP** - stack pointer.

AH	AL
BH	BL
CH	CL
DH	DL

SI (Source Idx)

DI (Dest. Idx)

BP (Base Ptr)

SP (Stack Ptr)

Z (Flag Reg)

CS (Code Seg Reg)

DS (Data Seg Reg)

ES (Extra Seg Reg)

SS (Stack Seg Reg)

IP (Intr Ptr)

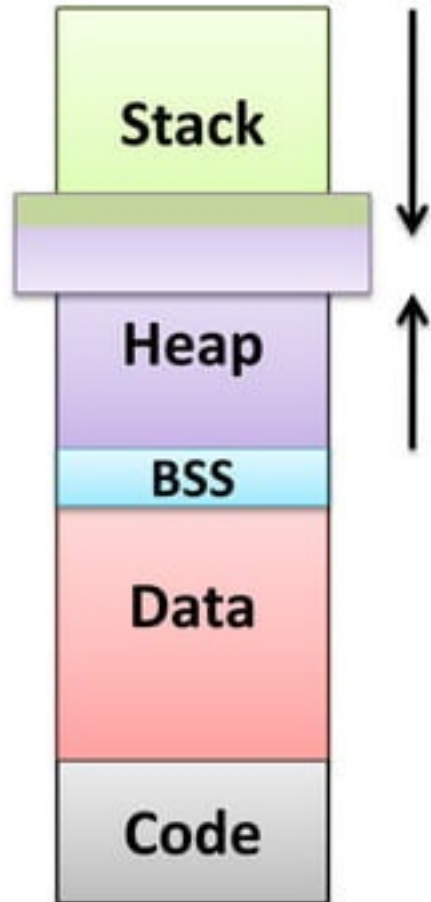
i386/i486/i686 Registers

31	15	7	0
EAX	AH	AL	
EBX	BH	BL	
ECX	CH	CL	
EDX	DH	DL	
ESI	SI (Source Idx)		
EDI	DI (Dest. Idx)		
EBP	BP (Base Ptr)		
ESP	SP (Stack Ptr)		
EZ	Z (Flag Reg)		
ECS	CS (Code Seg Reg)		
EDS	DS (Data Seg Reg)		
EES	ES (Extra Seg Reg)		
ESS	SS (Stack Seg Reg)		
EIP	IP (Intr Ptr)		

Extended

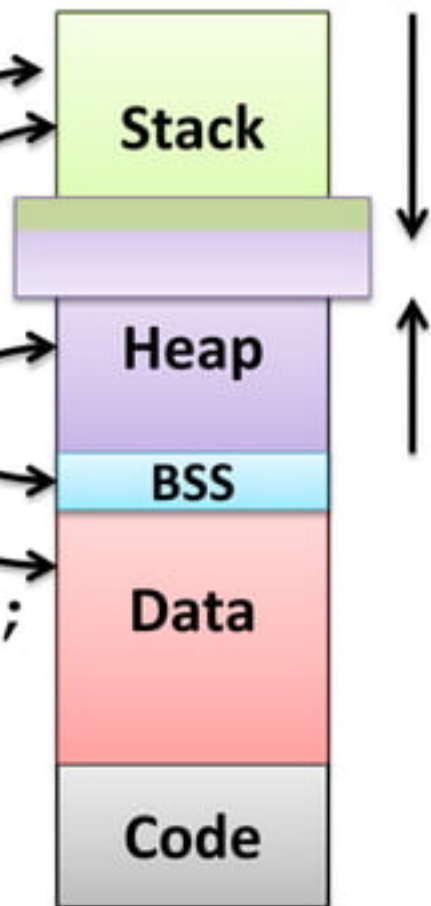
Memory layout of C program

- Stack
 - automatic (default), local
 - Initialized/uninitialized
- Data
 - Global, static, extern
 - BSS: Block Started by Symbol
 - BBS: Uninitialized Data Seg.
- Code
 - program instructions
- Heap
 - malloc, calloc



Memory layout of C program

```
int A;  
int B=10;  
main() {  
    int Alocal;  
    int *p;  
    p=(int*)malloc(10);  
}
```



MASM : Hello world

.model small

.stack 100h ; reserve 256 bytes of stack space

.data

message db "Hello world, I'm learning Assembly\$"

.code

main proc

mov ax, seg message

mov ds, ax

mov ah, 09 // 9 in the AH reg indicates Procedure
//should write a bit-string to the screen.

lea dx, message // Load Eff Address

int 21h

mov ax, 4c00h // Halt for DOS routine (Exit Program)

int 21h

main endp

end main

Memory Model: Segment Definition

- **.model small**
 - Most widely used memory model.
 - The code must fit in 64k.
 - The data must fit in 64k.
- **.model medium**
 - The code can exceed 64k.
 - The data must fit in 64k.
- **.model compact**
 - The code must fit in 64k.
 - The data can exceed 64k.
- **.medium and .compact are opposites.**

Data Allocation Directives

- **db :** define byte
- **dw:** def. word (2 bytes)
- **dd:** def double word (4)
- **dq :** def quad word (8)
- **equ :** equate assign numeric
- **expr** to a name

.data

db A 100 dup (?) ; define 100 bytes, with no initial values for bytes

db "Hello" ; define 5 bytes, ASCII equivalent of "Hello".

dd PtrArray 4 dup (?) ;array[0..3] of dword

maxint equ 32767 ; define maxint=32767

count equ 10 * 20 ; calculate a value (200)

MASM: Loop

- Assembly code: Loop
 - Loop simply decreases CX and checks if CX != 0, if so, a Jump to the specified memory location

```
MOV CX,100  
_LABEL: INC AX  
LOOP _LABEL
```

- LOOPNZ : LOOPS when the zero flag is not set

```
MOV CX,10  
_CMPLOOP: DEC AX  
CMP AX,3  
LOOPNE CMPLOOP
```

MASM: Nested Loop

- Assembly code: Nested Loop: One CX register

```
        mov     cx, 8
Loop1:  push    cx
        mov     cx, 4
Loop2:  stmts
        loop    Loop2
        pop     cx
        stmts
        loop    Loop1
```

Next Class Agenda

- Detail of Assembly language program
- Addressing mode
- Example program
- Assignment problem
- Summary of 8085/8085/i386 Arch & programming (1st Unit of Syllabus)
- Introduction to device interfacing
 - Device type
 - Interfacing

Thanks