



Trinity College Dublin

Coláiste na Tríonóide, Baile Átha Cliath

The University of Dublin

06 – Exceptions

CS1022 – Introduction to Computing II

Dr Adam Taylor / adam.taylor@tcd.ie

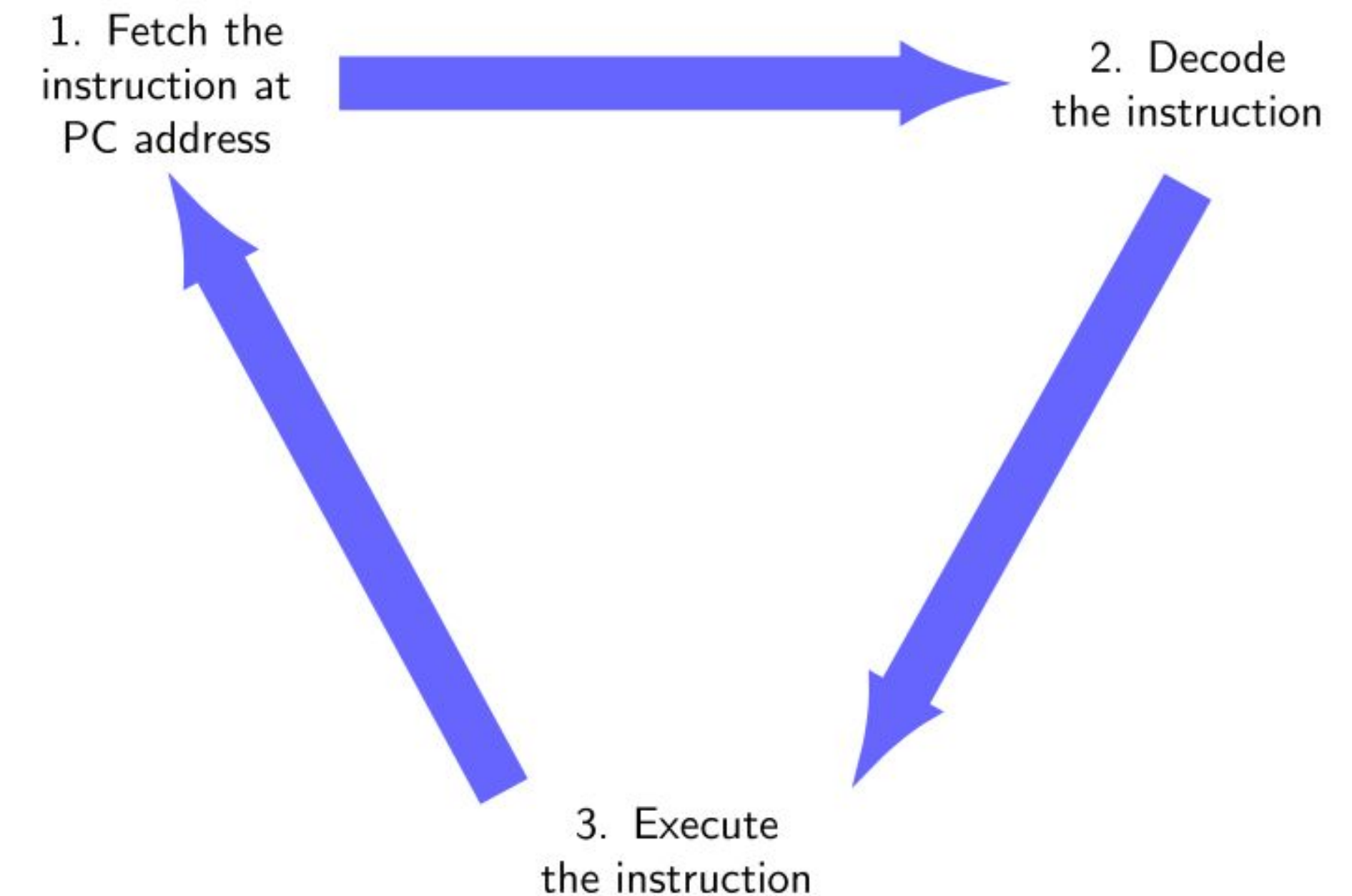
School of Computer Science and
Statistics

Exceptions – Events outside the normal (expected?) flow of execution of a program

Suppose the processor fetches the next instruction from memory but the instruction word is not a valid instruction

Suppose we try to write a word to address 0xA1000000 but the memory device at that address is a read-only ROM device

Suppose we're executing an instruction and someone presses the RESET button ...



Expected events that occur at unpredictable times (with respect to the execution of our program!)

Mouse clicks, keyboard presses, touchscreen presses, button presses, ...

Receive network data, finish sending network data, ...

Wait for a timer to count down to zero, ...

Unexpected events that we can try to handle

Eject CD, remove USB key, ...

Battery power low, ...

Loose wireless network signal, network cable unplugged, ...

Read from or write to invalid memory addresses

Attempting to execute invalid machine code

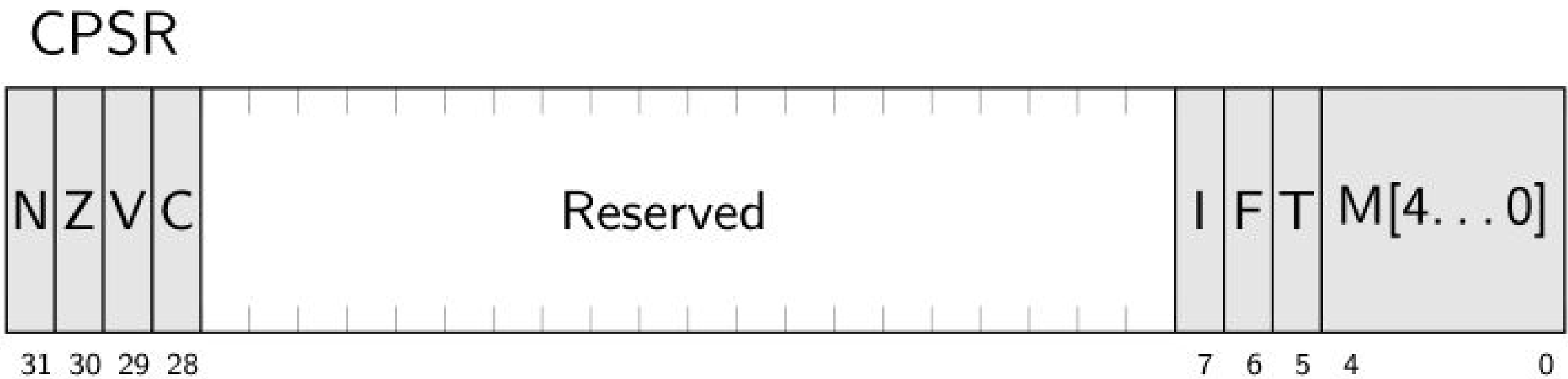
Reset

Name	Description
Reset	Occurs at power-on or when RESET button is pressed
Undefined Instruction	Attempt to execute an invalid instruction word
Software Interrupt (SWI)	Caused programmatically by executing SWI instruction
Prefetch Abort	Attempt to fetch an instruction from an invalid address (instructions)
Data Abort	Attempt to load/store from/to an invalid address (data)
(Reserved)	
IRQ	Interrupt ReQuest
FIQ	Fast Interrupt reQuest

When one of these exceptions occurs, it can be “handled” by an “exception handler” (*think subroutine!*) that takes appropriate action

Processor Mode		Mode #	Description
User	usr	0b10000	Normal program execution
FIQ	fiq	0b10001	Fast Interrupt reQuest handling (low overhead)
IRQ	irq	0b10010	General purpose interrupt handling
Supervisor	svc	0b10011	Protected mode for OS (Reset / SWI)
Abort	abt	0b10111	Prefetch / Data Abort (memory management)
Undefined	und	0b11011	Software emulation of unimplemented instructions

System sys 0b11111 Privileged mode using user-mode registers (OS)



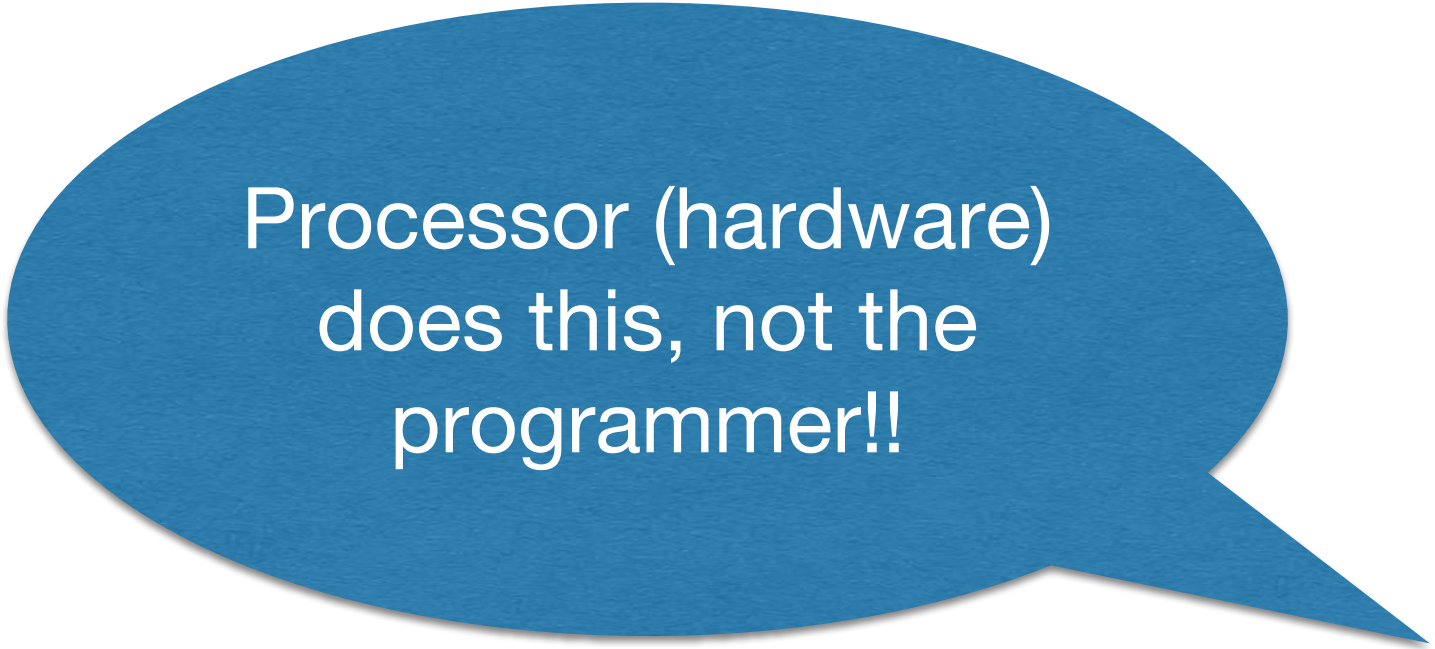
Modes						
<div><div>Privileged modes</div><div>Exception modes</div></div>						
User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	<div>R8_fiq</div>
R9	R9	R9	R9	R9	R9	<div>R9_fiq</div>
R10	R10	R10	R10	R10	R10	<div>R10_fiq</div>
R11	R11	R11	R11	R11	R11	<div>R11_fiq</div>
R12	R12	R12	R12	R12	R12	<div>R12_fiq</div>
R13	R13	<div>R13_svc</div>	<div>R13_abt</div>	<div>R13_und</div>	<div>R13_irq</div>	<div>R13_fiq</div>
R14	R14	<div>R14_svc</div>	<div>R14_abt</div>	<div>R14_und</div>	<div>R14_irq</div>	<div>R14_fiq</div>
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		<div>SPSR_svc</div>	<div>SPSR_abt</div>	<div>SPSR_und</div>	<div>SPSR_irq</div>	<div>SPSR_fiq</div>

indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode

ARM Programmers Model Registers

ARM Architecture Reference
Manual (Issue I), Figure A2-1

1. Complete execution of current instruction
2. Save current state and update CPSR
 - i. Save return address in LR_<mode>
(<mode> will be the new processor mode while handling the exception)
 - ii. CPSR saved to SPSR_<mode>
 - iii. Switch to ARM state (clear T bit)
(Exceptions are always handled in ARM state)
 - iv. Set mode (M[4...0] bits)
(Mode appropriate to exception type)
 - v. Disable IRQs (set I bit) for all exceptions
 - vi. Disable FIQs (set F bit) when handling FIQs and Reset
3. Change PC to address of the exception handler subroutine corresponding to the exception type



Processor (hardware)
does this, not the
programmer!!

Handler for each exception type starts at a fixed, pre-defined address

Processor loads PC with this address

How many instructions are there in each exception handler?

Address	Memory
	...
	Rest of memory
0x0000001C	FIQ exception handler
0x00000018	branch to IRQ exception handler
0x00000014	NOP
0x00000010	branch to DATA ABORT exception handler
0x0000000C	branch to PREFETCH ABORT exception handler
0x00000008	branch to SWI exception handler
0x00000004	branch to UNDEFINED IN-STRUCTION exception handler
0x00000000	branch to Reset exception handler

← 32 bits = 1 word = 4 bytes →

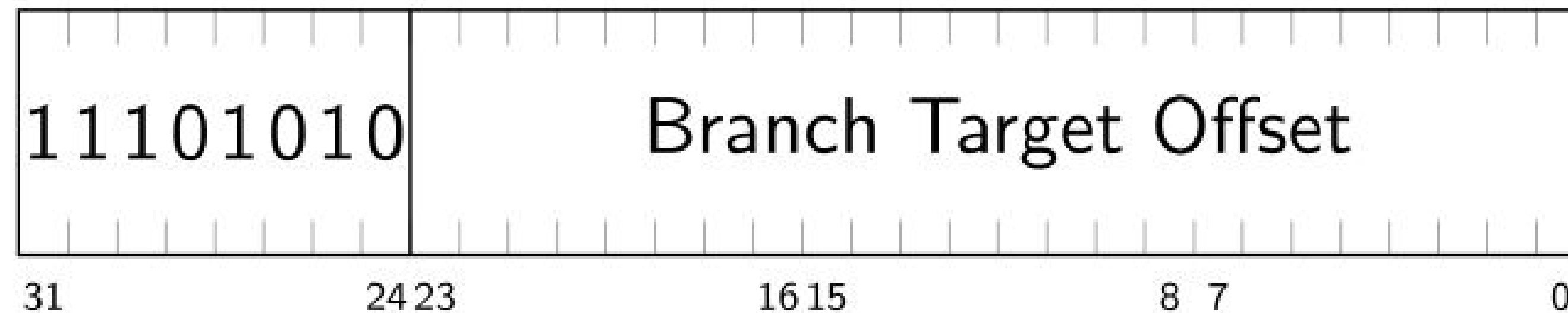
Eight words at the start of memory contain instructions that cause the processor to branch to the real exception handler for each exception type

This is the Exception Vector Table

Address	Memory
	...
	Rest of memory
0x0000001C	branch to FIQ handler
0x00000018	branch to IRQ handler
0x00000014	NOP
0x00000010	branch to DATA ABORT handler
0x0000000C	branch to PREFETCH handler
0x00000008	branch to SWI handler
0x00000004	branch to UNDEF-INSTR handler
0x00000000	branch to Reset handler
	...
	← 32 bits = 1 word = 4 bytes →

A number of ways we can branch to the start of the real exception handler ...

A branch (B) instruction – range limited to $\pm 32\text{MB}$



A MOV instruction (MOV PC,<addr>) – limited to jumping to an address that can be represented as a byte shifted by an even number of bits

An LDR (LDR PC,[offset]) instruction – loads start address of the real exception from PC+offset into the PC (but we need to store the real exception handler start addresses in memory)

	Rest of memory
EVT + 0x18	branch to IRQ handler
EVT + 0x14	NOP
EVT + 0x10	branch to DATA ABORT handler
EVT + 0x0C	branch to PREFETCH handler
EVT + 0x08	branch to SWI handler
EVT + 0x04	branch to UNDEF-INSTR handler
EVT + 0x00	branch to Reset handler
	...
0x0000001C	FIQ handler
0x00000018	branch to IRQ handler
0x00000014	NOP
0x00000010	branch to DATA ABORT handler
0x0000000C	branch to PREFETCH handler
0x00000008	branch to SWI handler
0x00000004	branch to UNDEF-INSTR handler
0x00000000	branch to Reset handler

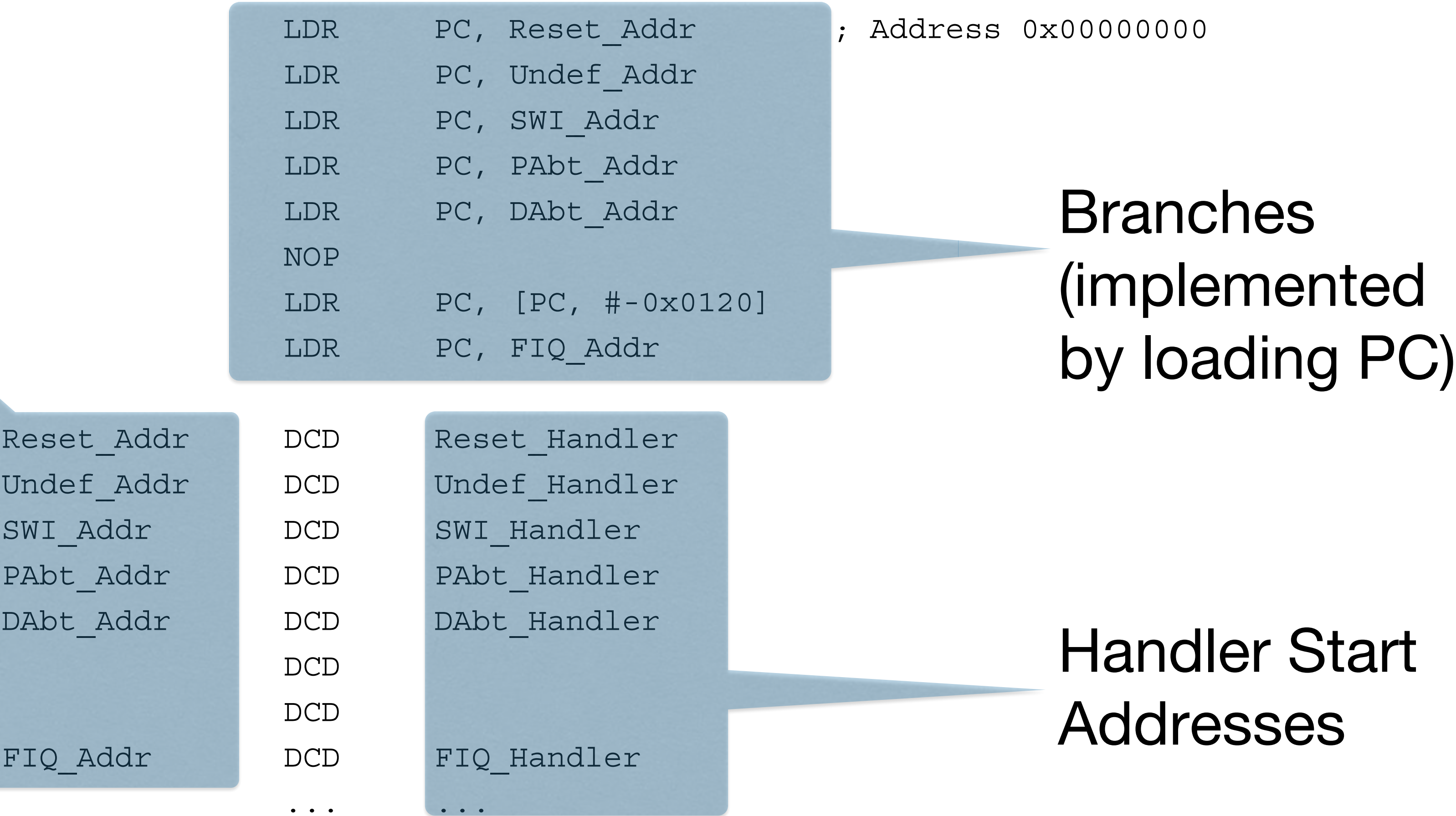
```
; e.g. branch to SWI handler
LDR PC, [EVT + 0x08]
```

```
; e.g. branch to reset handler
LDR PC, [EVT + 0x00]
```

Exception Vector Table

AREA RESET, CODE, READONLY

; Exception Vectors
; Mapped to Address 0.
; Absolute addressing mode must be used.
; Dummy Handlers are implemented as infinite loops which can be modified.



Must always be a handler for each exception, even if it does nothing ...

```
Undef_Handler    B    Undef_Handler
SWI_Handler      B    SWI_Handler
PAbt_Handler     B    PAbt_Handler
DAbt_Handler     B    DAbt_Handler
IRQ_Handler      B    IRQ_Handler
FIQ_Handler      B    FIQ_Handler
```

```
; Reset Handler
```

```
Reset_Handler
```

```
    < reset handler code goes here >
```

IRQ handler address is specified differently (more later)

FIQ is a special case and is often implemented differently (designed to reduce overheads and allow faster exception handling)

Always have a RESET handler to perform system initialisation

ARM instructions are 32-bits long

2^{32} possible instruction words

Not all instruction words are valid

Invalid instructions raise undefined instruction exceptions

Take advantage of this to extend the instruction set with our own instructions (*instruction emulation*)

Instruction operation must be implemented in software

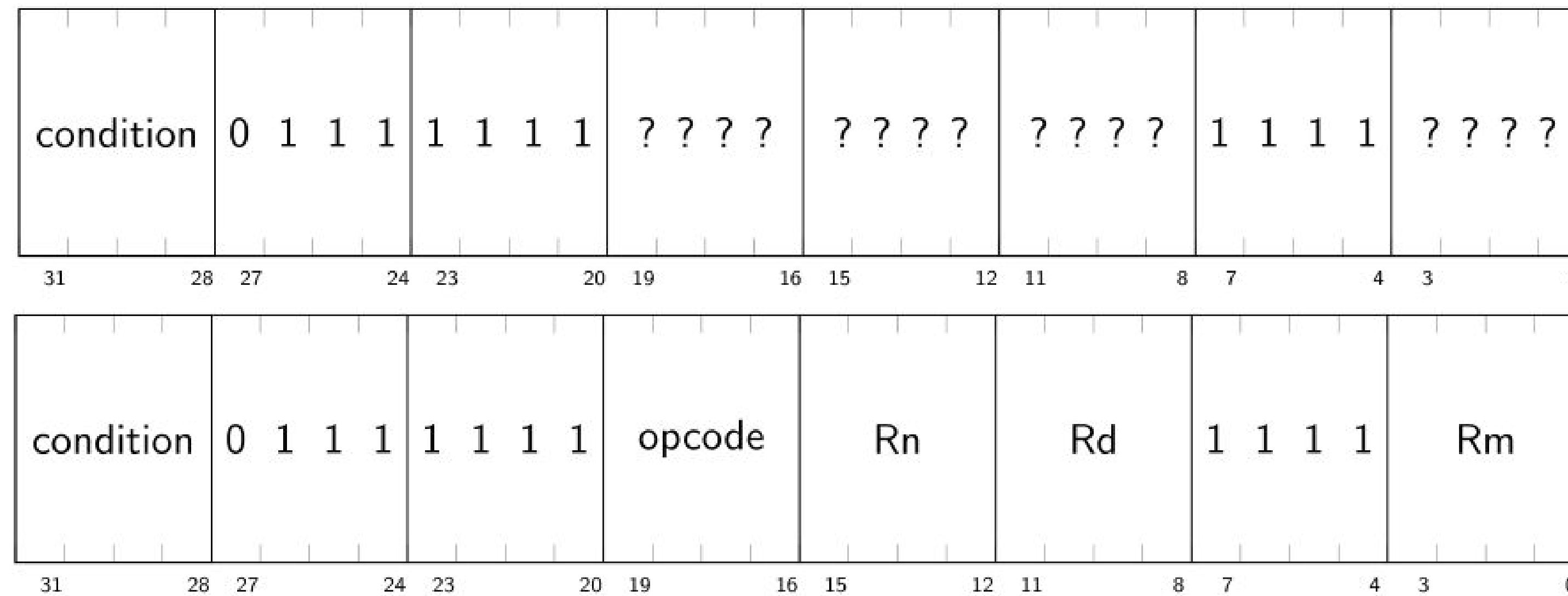
When the processor attempts to execute it, an undefined instruction exception is raised

Undefined instruction exception handler is executed

We provide our own undefined instruction exception handler to decode the instruction and implement the desired operation

Provide a POWER instruction to compute x^y

Define our instruction template



undefined
instruction
space

$Rd =$
 Rm^{Rn}

1. Write our Undefined exception handler
2. Set up the vector table
3. Test the instruction

Example – Undefined POWER instruction

16

UndefHandler

```
STMFD sp!, {r0-r12, LR} ; save registers
```

```
LDR r4, [lr, #-4] ; load undefined instruction
```

```
BIC r5, r4, #0xFFF0FFFF ; clear all but opcode bits
```

```
TEQ r5, #0x00010000 ; check for undefined opcode 0x1
```

```
BNE endif1 ; if (power instruction) {
```

```
BIC r5, r4, #0xFFFFFFFF0 ; isolate Rm register number
```

```
BIC r6, r4, #0xFFFF0FFF ; isolate Rn register number
```

```
MOV r6, r6, LSR #12 ;
```

```
BIC r7, r4, #0xFFFFF0FF ; isolate Rd register number
```

```
MOV r7, r7, LSR #8 ;
```

```
LDR r1, [sp, r5, LSL #2] ; read saved Rm from stack (don't pop)
```

```
LDR r2, [sp, r6, LSL #2] ; read saved Rn from stack (don't pop)
```

```
BLpower ; call pow subroutine
```

```
STR r0, [sp, r7, LSL #2] ; save result over saved Rd
```

```
endif1 ; }
```

```
LDMFD sp!, {r0-r12, PC}^ ; restore register and CPSR
```

```
;
; Install address of Exception Handler in table
;
LDR r4, =0x40000024; 0x00000024 is mapped to 0x40000024!
LDR r5, =UndefHandler ; Address of new undefined handler
STR r5, [r4] ; Store new undef handler address in table

;
; Test our new instruction
;
LDR r4, =3 ; test 3^4
LDR r5, =4 ;

; This is our undefined instruction opcode
DCD 0x77F150F4 ; POW r0, r4, r5 (r0 = r4 ^ r5)

; R0 should be 81 now!
```

Example – Undefined POWER instruction

18

```
; power subroutine
; Computes x^y
; paramaters: r1: x (value)
;   r2: y (value)
; return:  r0: result (variable)
power
    STMFD  sp!, {r1-r2,lr} ; save registers

    CMP  r2, #0 ; if (y = 0)
    BNE  else2  ; {
    MOV  r0, #1 ; result = 1
    B  endif2 ; }
else2  ; else {
    MOV  r0, r1 ; result = x
    SUBS r2, r2, #1 ; y = y - 1
    BEQ  endif3 ; if (y != 0) {
do4      ; do {
    MUL  r0, r1, r0 ; result = result * x
    SUBS r2, r2, #1 ; y = y - 1
    BNE  do4  ; } while (y != 0)
endif3      ; }
endif2      ; }
    LDMFD  sp!, {r1-r2, pc} ; restore registers and return
```