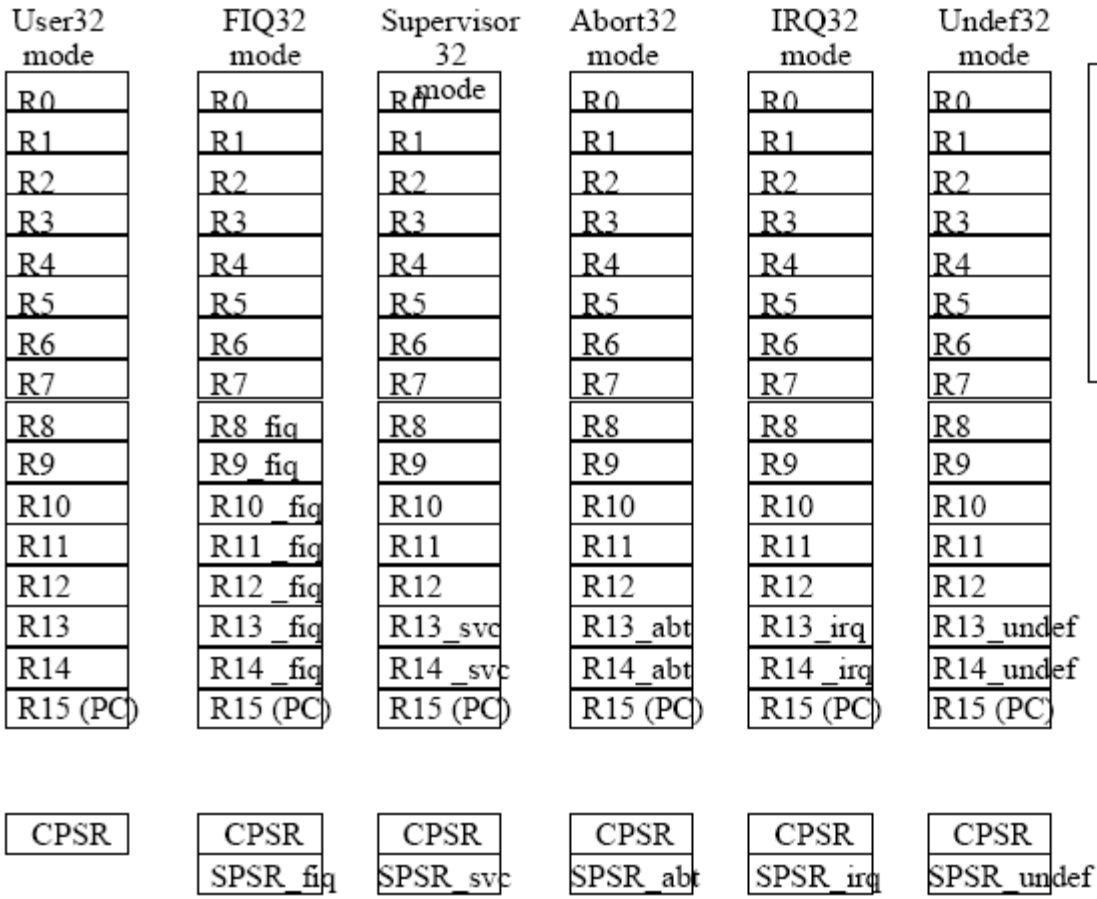
ARM

Exception and Interrupt Handler

Normal Program Flow vs. Exceptions

- Normally, programs execute sequentially (with a few branches to make life interesting)
- Normally, programs execute in user mode (see next slide)
- **Exceptions** and **interrupts** break the sequential flow of a program, jumping to architecturally-defined memory locations
- In ARM, SoftWare Interrupt (SWI) is the "system call" exception
- Types of ARM exceptions
  - reset — when CPU reset pin is asserted
  - undefined instruction — when CPU tries to execute an undefined opcode
  - software interrupt — when CPU executes the SWI instruction
  - prefetch abort — when CPU tries to execute an instruction prefetched from an illegal address
  - Data abort — when data transfer instruction tries to read or write at an illegal address
  - IRQ — when CPU's external interrupt request pin is asserted
  - FIQ — when CPU's external fast interrupt request pin is asserted

Different ARM Modes



- R14 is hardwired to be the link register
- R15 is hardwired to be the PC
- R13, by convention, is the stack pointer

Terminology

- The terms exception and interrupt are often confused
- Exception usually refers to an internal CPU event such as
  - floating point overflow
  - MMU fault (e.g., page fault)
  - trap (SWI)
- Interrupt usually refers to an external I/O event such as
  - I/O device request
  - reset
- In the ARM architecture manuals, the two terms are mixed together

Exceptions, Interrupts, and the Vector Table

- When an exception or interrupt occurs, the processor set the PC to a specific memory address
- The address is within a special address range called the vector table
- The entries in the vector table are instructions that branch to specific routines designed to handle a particular exception or interrupt
- When an exception or interrupt occurs,the processor suspends normal execution and starts loading instructions from the exception vector table.

Exceptions, Interrupts, and the Vector Table

Exception / Interrupt	Shorthand	Address	High Address
Reset	RESET	0x00000000	0xffff0000
Undefined Instruction	UNDEF	0x00000004	0xffff0004
Software Interrupt	SWI	0x00000008	0xffff0008
Prefetch Abort	PABT	0x0000000C	0xffff000C
Data Abort	DABT	0x000000010	0xffff0010
Reserved	-	0x000000014	0xffff0014
Interrupt Request	IRQ	0x000000018	0xffff0018
Fast Interrupt Request	FIQ	0x00000001C	0xffff001C

Exceptions, Interrupts, and the Vector Table

- **RESET** – when power is applied, branches to initialization code
- **UNDEF** – when the processor cannot decode an instruction
- **SWI** – when a SWI instruction is called
- **PABT** – attempts to fetch an instruction from an address without the correct access permissions
- **DABT** –attempts to access data memory without the correct access permissions
- **IRQ** – by external hardware
- **FIQ** – by external hardware requiring faster response time

Exception Handling

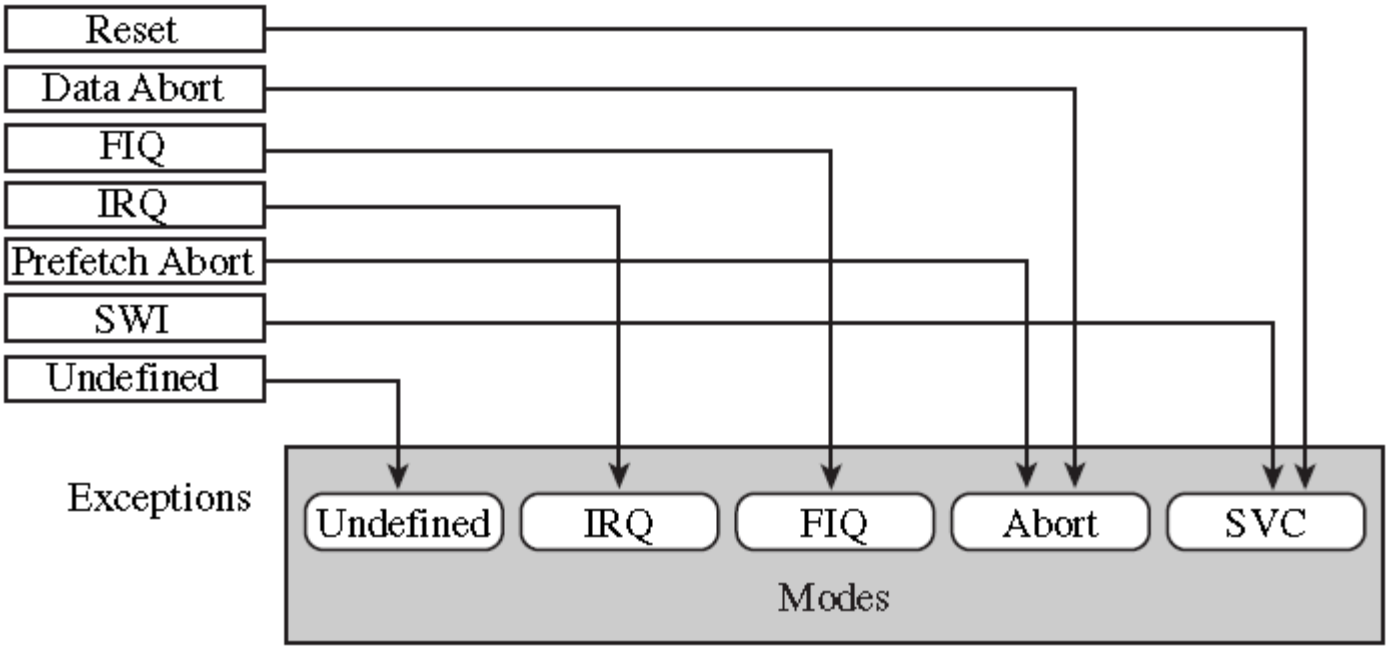
- ARM processor modes and Exceptions
- Vector table
- Exception Priorities
- Link Register Offsets

ARM processor modes and Exceptions

- When an exception causes the mode change, the core automatically
  - Save CPSR to SPSR
  - PC to LR
  - Sets cpsr to exception mode
  - Set PC to address of exception handler

Exception	Mode	Main purpose
Fast Interrupt Request	*FIQ*	fast interrupt request handling
Interrupt Request	*IRQ*	interrupt request handling
SWI and Reset	*SVC*	protected mode for operating systems
Prefetch Abort and Data Abort	*abort*	virtual memory and/or memory protection handling
Undefined Instruction	*undefined*	software emulation of hardware coprocessors

- When an exception occurs ARM processor always switches to ARM STATE.



Vector table

- B<address>
- LDR pc, [pc, #offset]
- MOV pc, #immediate

Address	Exception	Mode on entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

At these addresses we find a jump instruction like that:

ldr pc, [pc, # IRQ handler offset]

priorities

Exceptions	Priority	*I* bit	*F* bit
Reset	1	1	1
Data Abort	2	1	—
Fast Interrupt Request	3	1	1
Interrupt Request	4	1	—
Prefetch Abort	5	1	—
Software Interrupt	6	1	—
Undefined Instruction	6	1	—

- Link register offset

Here are the steps that the ARM processor does to handle an exception [5]:

- Preserve the address of the next instruction.
- Copy ***CPSR*** to the appropriate ***SPSR***, which is one of the banked registers for each mode of operation.
- Force the ***CPSR*** mode bits to a value depending on the raised exception.
- Force the ***PC*** to fetch the next instruction from the exception vector table.
- Now the handler is running in the mode associated with the raised exception.
- When handler is done, the ***CPSR*** is restored from the saved ***SPSR.***
- ***PC*** is updated with the value of (***LR*** – offset) and the offset value depends on the type of the exception.

And when deciding to leave the exception handler, the following steps occurs:

- Move the Link Register ***LR*** (minus an offset) to the ***PC***.
- Copy ***SPSR*** back to ***CPSR,*** this will automatically changes the mode back to the previous one.
- Clear the interrupt disable flags (if they were set).

Interrupts

- Two types of interrupts
  - External
  - SWI

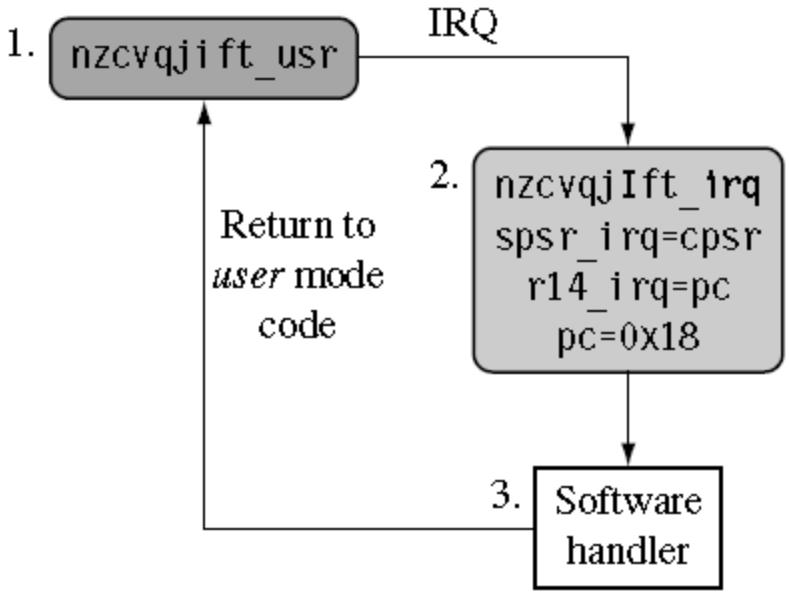
Interrupt Latency

It is the interval of time between from an external interrupt signal being raised to the first fetch of an instruction of the ISR of the raised interrupt signal.

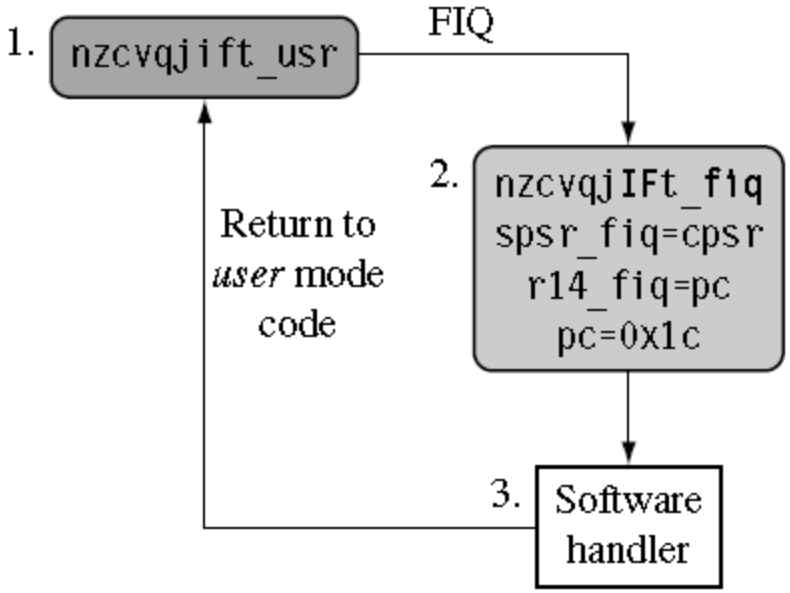
- Interrupt latency can be minimized by two methods
  - Nested Interrupt-system can respond to new interrupts while handling the old
  - Giving priorities to different interrupt sources

IRQ and FIQ Exception

- The ARM processor will continue executing the current instruction in the pipeline before handling the interrupt.
- The processor hardware go through the following standard procedure
  - The processor changes to a specific mode depending on the received interrupt.
  - The previous mode ***CPSR*** is saved in ***SPSR*** of the new mode.
  - The ***PC*** is saved in the ***LR*** of the new mode.
  - Either IRQ or both IRQ and FIQ exceptions are disabled in the cpsr.
  - The processor branches to a specific entry in the vector table.


1.
nzcvqjift_usr
IRQ
2.
nzcvqjIft_irq
spsr_irq=cpsr
r14_irq=pc
pc=0x18
Return to
user mode
code
3.
Software
handler

IRQ


1.
nzcvqjift_usr
FIQ
2.
nzcvqjIFt_fiq
spsr_fiq=cpsr
r14_fiq=pc
pc=0x1c
Return to
user mode
code
3.
Software
handler

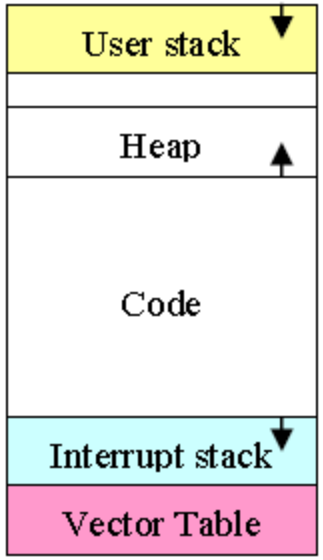
FIQ

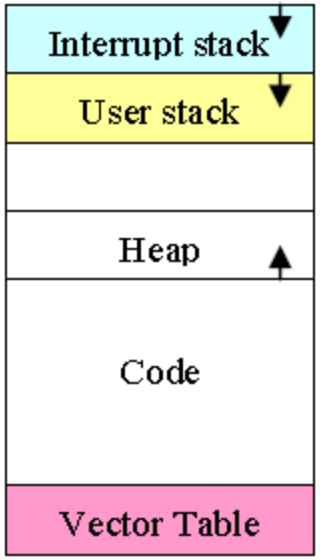
Stack Organization

- Stack area is different for each and every mode
- Descending Stack is common
- Nested interrupt handler requires larger stack
- While choosing stack location Should not overrun the vector table

Interrupt Stack

- Context switching
- Each mode has a dedicated register containing a stack pointer





Exception handlers

- Reset Handler
  - Initializes the system, setting Stack pointers, memory, External interrupt sources before enabling the FIQ and IRQ.
  - Code should be designed to avoid further triggering of exceptions

- Data Abort
  - FIQ can be raised within DATA abort handler
- FIQ
  - External peripheral generates FIQ signal
  - Core disables both FIQ and IRQ

- IRQ
  - Occurs when an External device generates IRQ signal
  - IRQ handler will be entered if neither an FIQ and Data abort occurs.
  - On entry IRQ exception is disabled and should remain disabled for the handler if not enabled by the handler

- Prefetch abort
  - Occurs when an attempt to fetch an instruction results in memory fault.
  - FIQ exception can be serviced
- Undefined instruction
  - Occurs when an instruction is not in ARM or thumb instruction
- SWI and undefined instruction have the same level of priority

- FIQ and IRQ can be serviced after the Execution stage
- Others can be serviced during execution stage.

Returning from exception handler

- Exception handler must not corrupt lr
- Moving correct value of lr into PC
- Restoring CPSR from SPSR

- For interrupt ,
  - When accessing r15[PC], r15=address of current instruction +8
  - Proper adjustment of LR is required while returning from handler

Interrupt Assignment

- An interrupt controller connects multiple external interrupts to either FIQ or IRQ
- IRQ are normally assigned to general purpose interrupts
  - Ex: periodic timer interrupt to force a context switch
- FIQ is reserved for an interrupt source which requires fast response time [less latency]

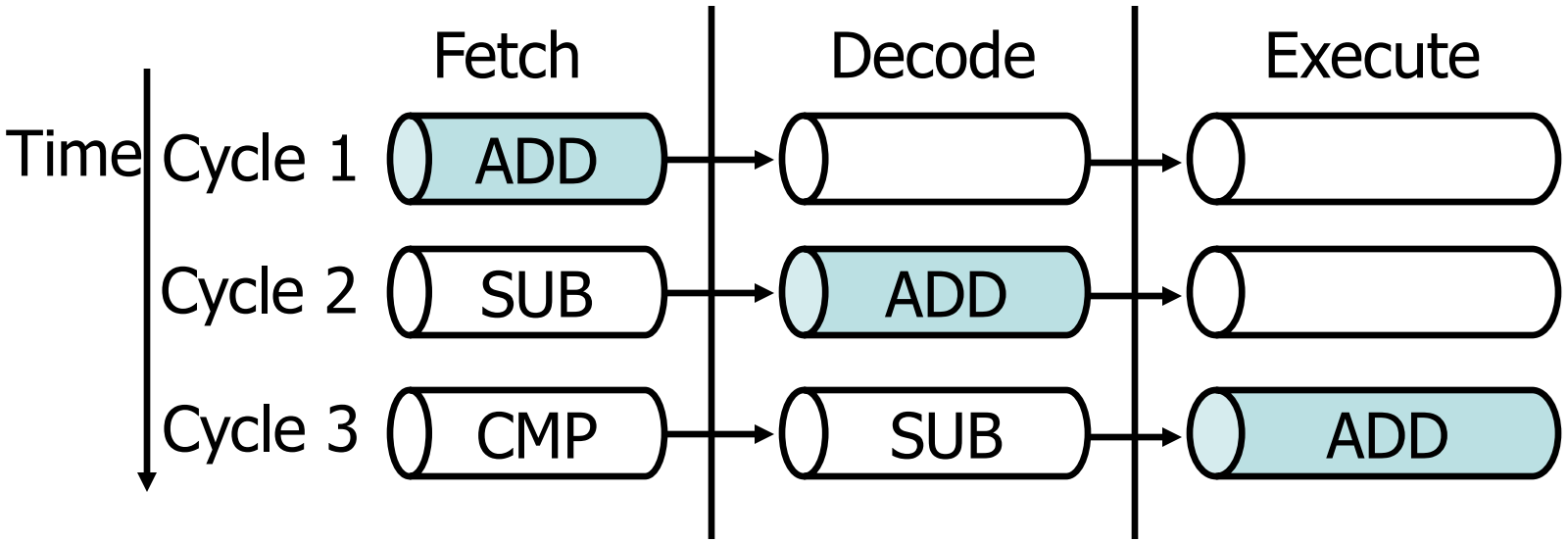
Interrupt Latency

- Hardware and software latency
  - Eg for HW is FIQ
  - Eg for SW is Nested handler, priority assignment, Average latency of Higher priority interrupt is less.

3 stage pipeline

- At any time slice, 3 different instruction may occupy each of these stages, so the hardware in each stage has to be capable of independent operations
- When the processor is executing data processing instructions, Latency =3 cycles
- When accessing R15, r15 = address of current instruction +8

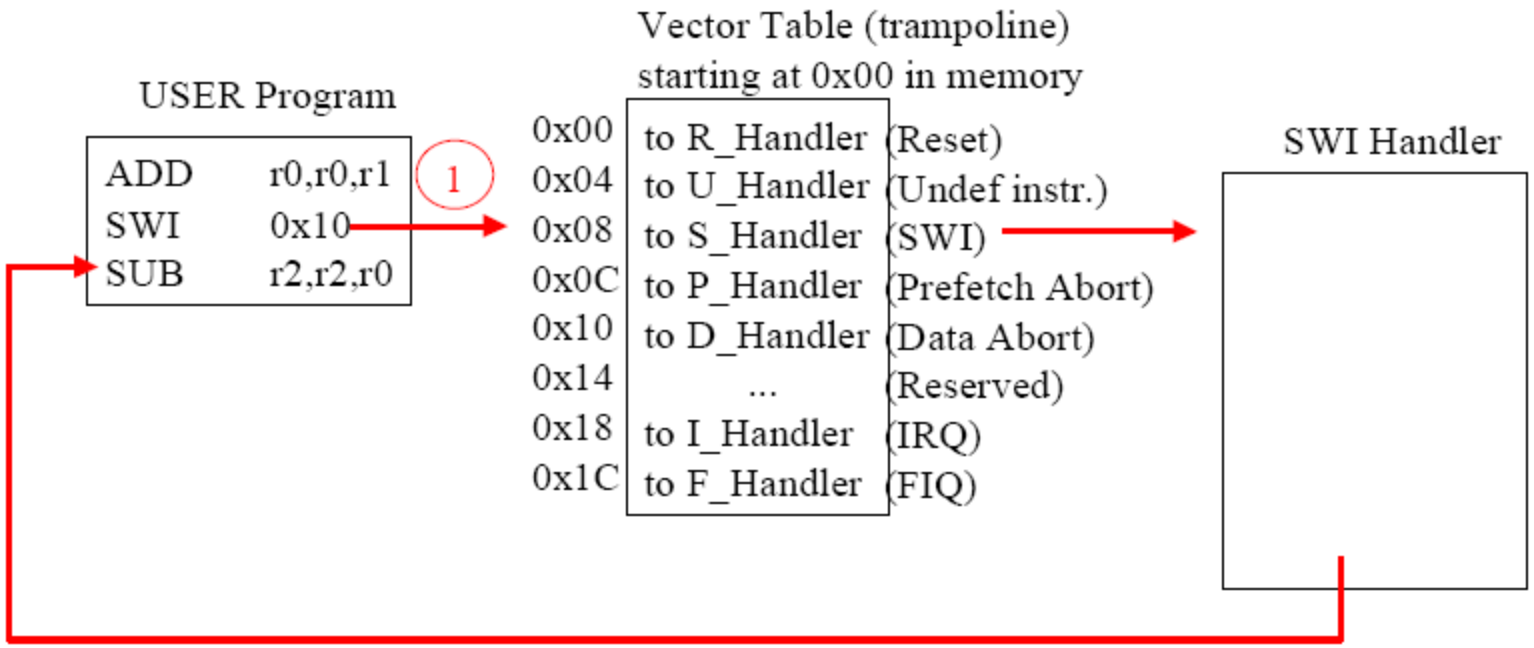
Pipelined instruction sequence



- Filling the pipeline
- Allows the core to execute an instruction every cycle

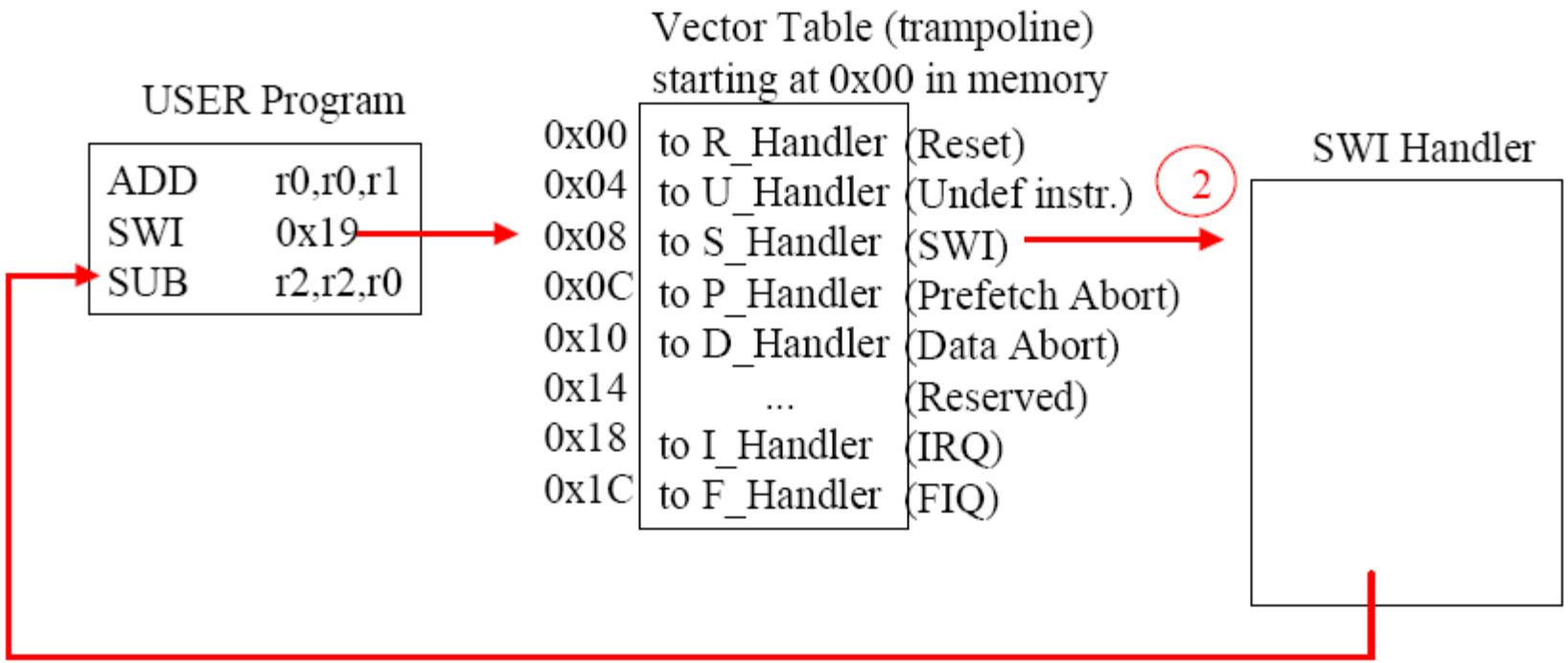
What Happens on an SWI

- ARM architecture defines Vector Table indexed by exception type
- For SWI, processor does: PC <--0x08 (1)
- Also, sets LR_svc, SPSR_svc, CPSR (supervisor mode, no IRQ)



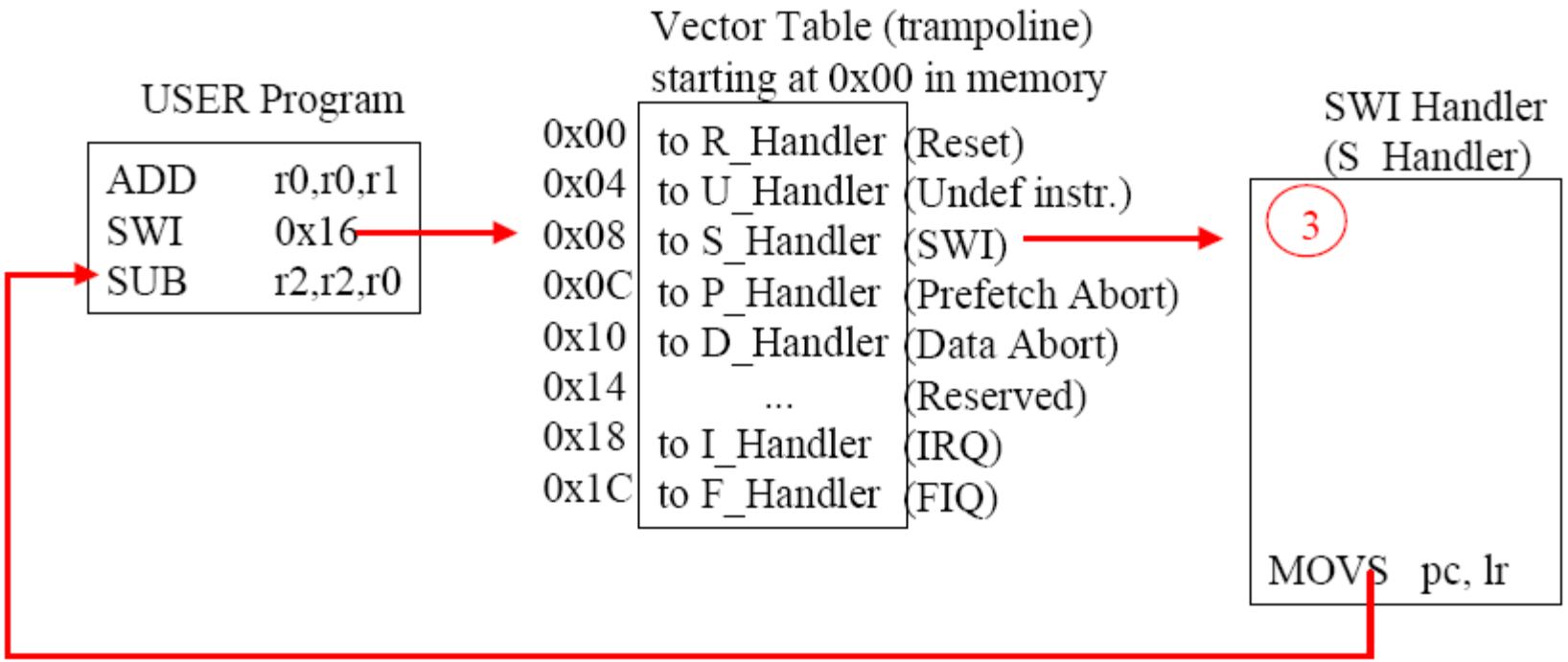
What happens on SWI (con't)

- Not enough space in the table (only one instruction per entry) to hold all of the code for the SWI handler function
- The one instruction must transfer control to SWI Handler (2)
- Several options (next slide)



What Happens on SWI (con't)

- Vectoring to the S_Handler starts executing the SWI handler
- When the handler is done, it returns to the program -- at the instruction following the SWI
- MOVS restores the original CPSR as well as changing pc



What do SWIs do?

- SWIs (often called software traps) allow a user program to call the OS -- that is, SWIs are how system calls are implemented
- When SWIs execute, the processor changes modes (from User to Supervisor on the ARM) and disables interrupts
- Types of SWIs in ARM Demon
  - SWI_WriteC (SWI 0) — Write a byte to the debug channel
  - SWI_Write0(SWI 2) — Write the null-terminated string to debug channel
  - SWI_ReadC(SWI 4) — Read a byte from the debug channel
  - SWI_Exit (SWI 0x11) — Halt emulation - this is how a program exits
  - SWI_EnterOS (SWI 0x16) — Put the processor in supervisor mode
  - SWI_Clock (SWI 0x61) — Return the number of centi-seconds
  - SWI_Time (SWI 0x63) — Return the number of secs since Jan. 1, 1970

SWI Handler

```
;
STMFD   sp!, {r0-r12, lr}

; Read the SWI instruction
LDR       r10, [lr, #-4]

; Mask off top 8 bits
BIC       r10, r10, #0xff000000

; r10 - contains the SWI number
BL        service_routine

; return from SWI handler
LDMFD   sp!, {r0-r12, pc}^
```