Embedded Systems Design

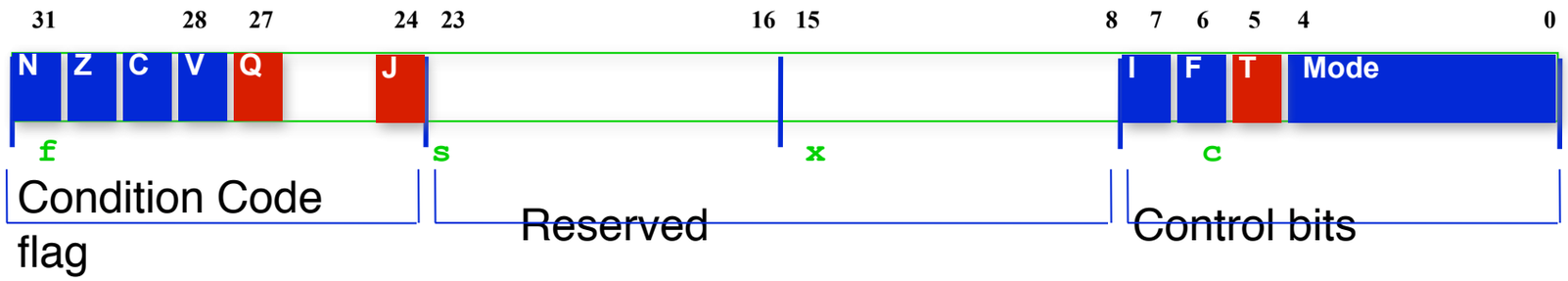
Lecture 6

Yongsoo Joo

KESL
Kookmin Univ. Embedded System Lab

Program Status Register

- PSR Registers in ARM
  - 1 CPSR
  - 5 SPSR (for each operating mode)
- PSR Register Fields
  - Condition flag
    - Reflects ALU processing results
  - Control bits
    - To control CPU status



PSR Register - flag bits

- Flag bits shows the results of the immediately preceding ALU calculation
  - 'N' bit: Negative flag
    - ALU calculation generates negative values
  - 'Z' bit: Zero flag
    - ALU calculation generates zero
  - 'C' bit: Carry flag
    - ALU calculation or shift operation generates carry
  - 'V' bit: Overflow bit
- Additional bits
  - 'Q' (saturation bit) and 'J' (Jazelle state) bits

PSR Register - control bits

- PSR control bits change processor operation modes, interrupts, and other processor states
  - I/F bit
    - Enable/disable IRQ or FIQ
  - T bit
    - Thumb mode on/off
    - Only controlled through 'BX' instruction
  - Mode bits

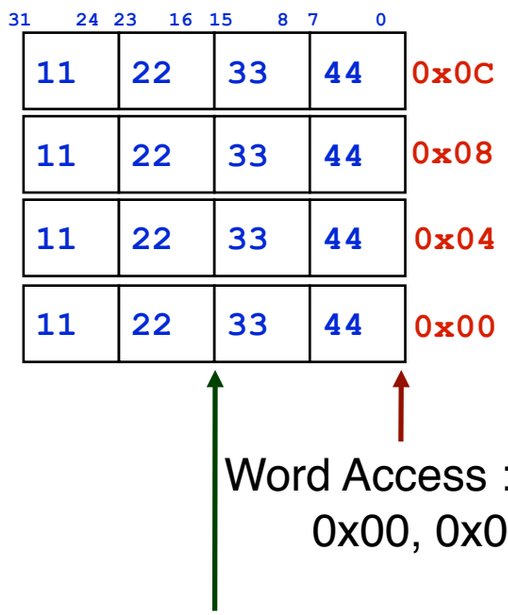
M[4:0]	Operating Mode	M[4:0]	Operating Mode
10000	User	10111	Abort
10001	FIQ	11011	Undefined
10010	IRQ	11111	System
10011	SVC		

Memory Architecture

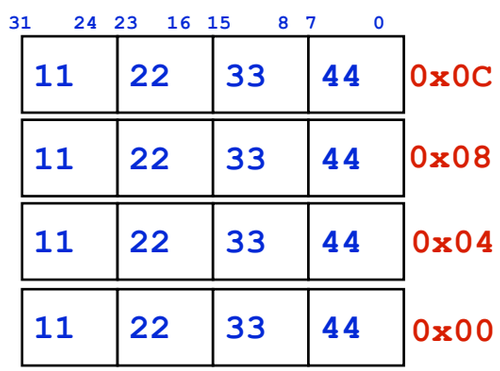
- Endian
  - Big/Little endian (ARM supports both)
  - Configured through hardware setting
- Data access types
  - Byte: 8 bits
  - Halfword: 16 bits
  - Word: 32 bits (in 32 bits processors)
- Aligned/Un-aligned access
  - All the memory accesses should be aligned by word(half-word)
  - Modern architecture allows un-aligned accesses
    - Previously always data aborts occur

Aligned and Un-aligned Accesses

Aligned Access



Un-aligned Access

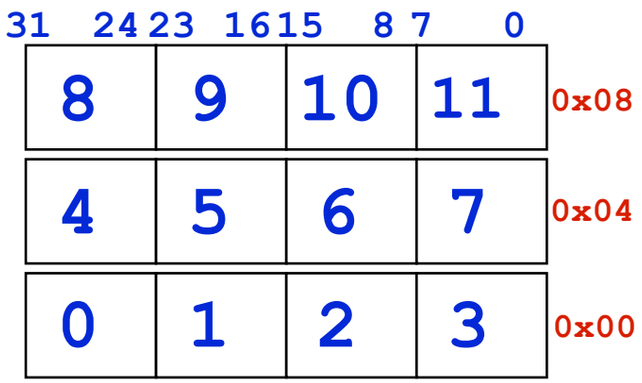


Word Access : Abort
0x01, 0x06, 0x07, …

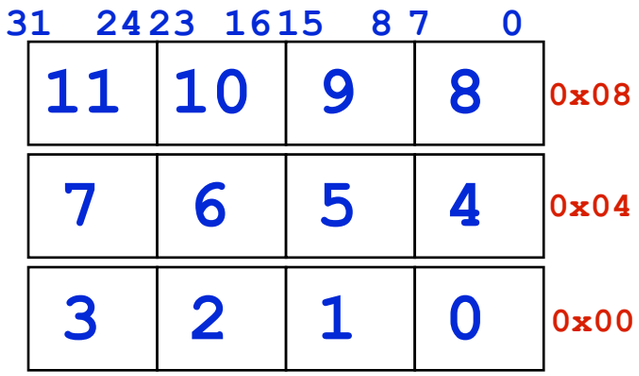
Half-Word Access : Abort
0x01, 0x03, 0x05 …

Little-Endian vs. Big-Endian

- Big-Endian
  - MSB (Most Significant Byte) is located in lower addresses in memory (Bits 24 ~ 31 are MSB)
- Little-Endian
  - LSB (Least Significant Byte) is located in lower addresses in memory (Bits 0 ~ 7 are LSB)



Big-Endian



Little-Endian

Little-Endian vs. Big-Endian

Table 7.108. Little-endian write, 64-bit external bus

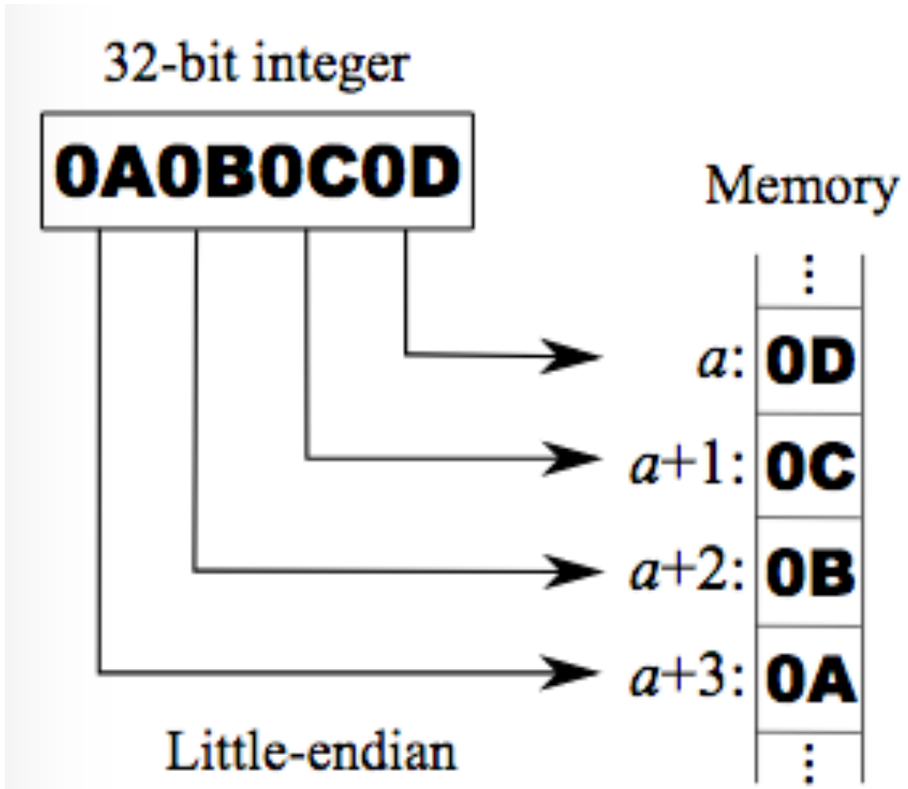
Internal transfer width	Access: Write, little-endian, 64-bit external bus		System data mapping on to external data bus MPMCDATA to HRDATA							
	HSIZE [2:0]	HADDR [2:0]	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
Byte	000	011	-	-	-	-	[31:24]		-	-
Byte	000	010	-	-	-	-	-	[23:16]	-	-
Byte	000	001	-	-	-	-	-	-	[15:8]	-
Byte	000	000	-	-	-	-	-	-	-	[7:0]

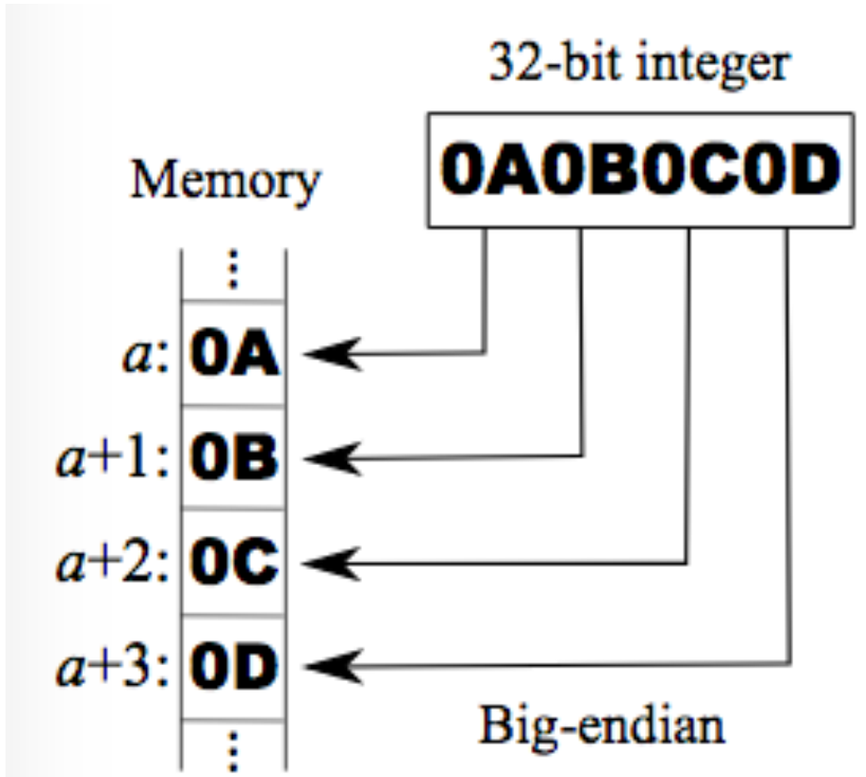
Table 7.109. Big-endian write, 64-bit external bus

Internal transfer width	Access: Write, big-endian, 64-bit external bus		System data mapping on to external data bus MPMCDATA to HRDATA							
	HSIZE [2:0]	HADDR [2:0]	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
Byte	000	011	-	-	-	-	-	-	-	[7:0]
Byte	000	010	-	-	-	-	-	-	[15:8]	-
Byte	000	001	-	-	-	-	-	[23:16]	-	-
Byte	000	000	-	-	-	-	[31:24]	-	-	-

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0269a/CCHIBGCH.html

Little-Endian vs. Big-Endian





https://en.wikipedia.org/wiki/Endianness

Exceptions

- Exceptions
  - Any interruption in regular control-flow execution caused by external requests, errors, and faults
  - Examples: external interrupts
- ARM Exceptions
  - Reset
  - Undefined instruction
  - Software interrupt
  - Prefetch abort
  - Data abort
  - IRQ
  - FIQ

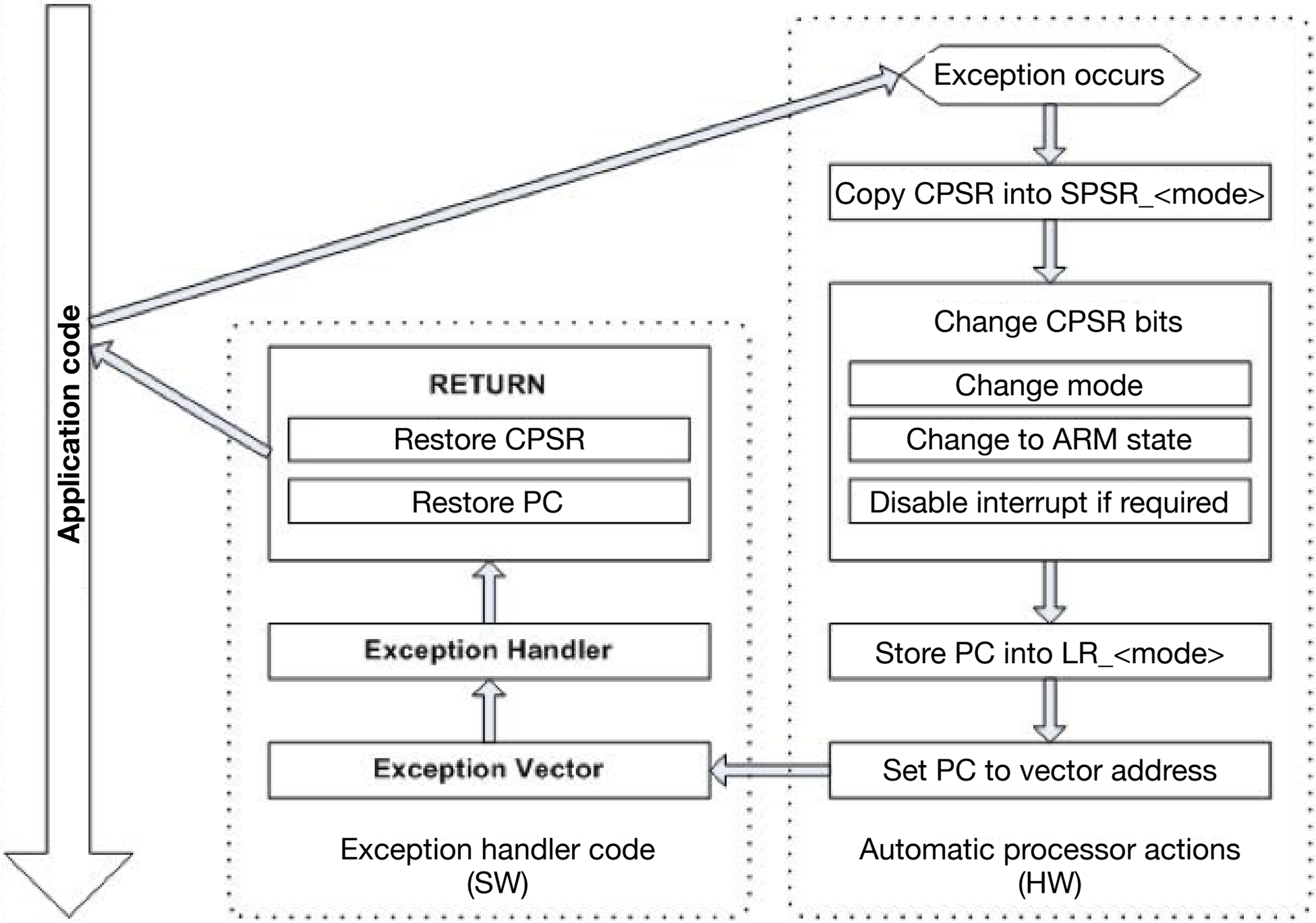
Exception Vectors

- Exception vector and Exception vector table
  - The pre-determined address to handle each exception
  - The table of exception handler addresses for all the exceptions
  - A fixed location in memory is allocated to exception vector table
  - Each entry of exception vector table is JUMP/BRANCH instruction to each exception handler
  - Default exception vector table location is 0x00000000

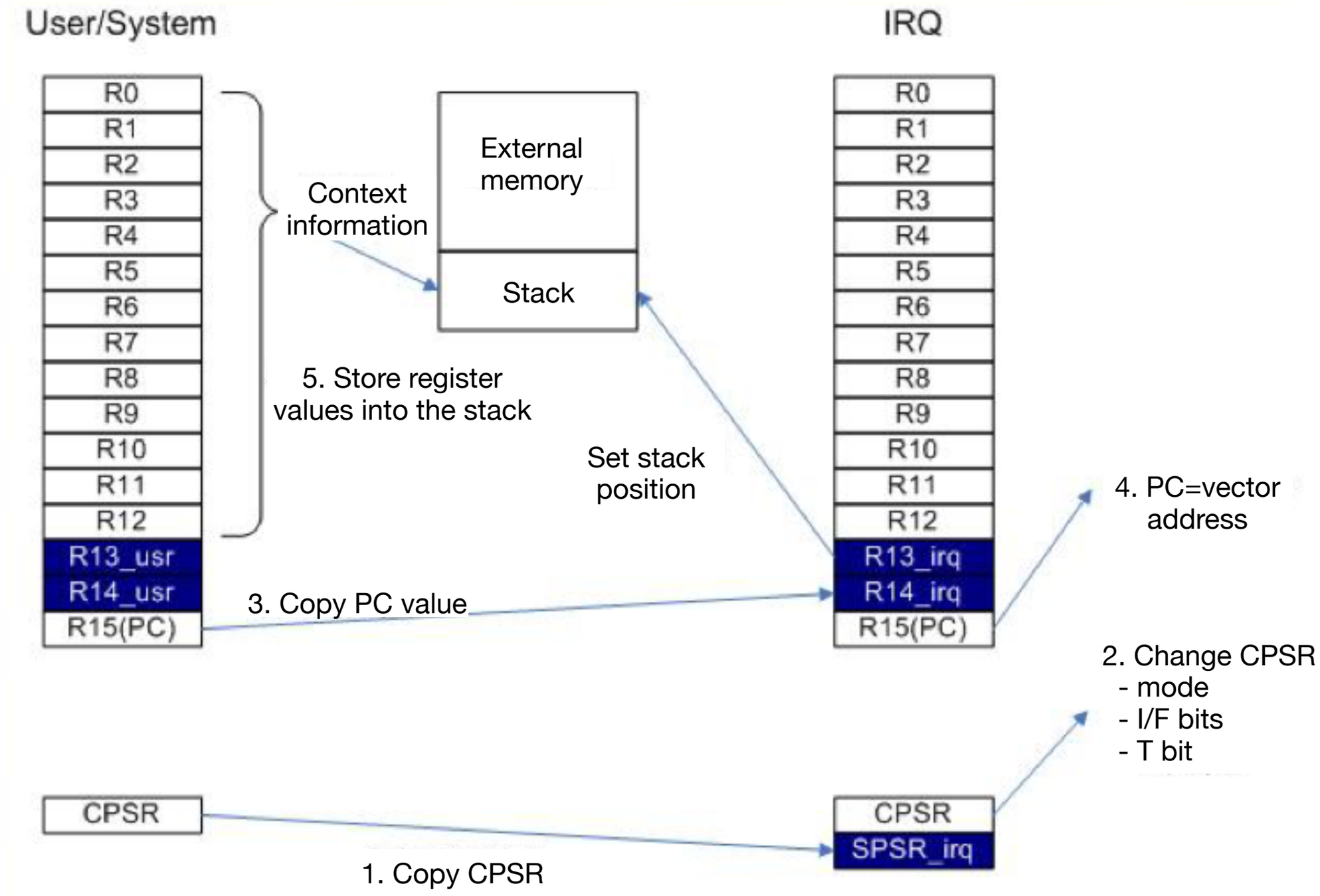
Exception Vector Table

Exception	Vector Address	Prioriry	Operation mode
Reset	**0x0000 0000**	**1 (High)**	**Supervisor(SVC)**
Undefined Instruction	**0x0000 0004**	**6 (Low)**	**Undefined**
Software Interrupt(SWI)	**0x0000 0008**	**6**	**Supervisor(SVC)**
Prefetch Abort	**0x0000 000C**	**5**	**Abort**
Data Abort	**0x0000 0010**	**2**	**Abort**
Reserved	**0x0000 0014**		
IRQ	**0x0000 0018**	**4**	**IRQ**
FIQ	**0x0000 001C**	**3**	**FIQ**

Exception Handling



Exception Handling



32 Bits ARM Instruction Format

31	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Cond		0	0	I	Opcode				S	Rn				Rd				Operand 2												
Cond		0	0	0	0	0	0	A	S	Rd				Rn				Rs		1	0	0	1	Rm					
Cond		0	0	0	0	1	U	A	S	RdHi				RdLo				Rs		1	0	0	1	Rm					
Cond		0	0	0	1	0	B	0	0	Rn				Rd				0	0	0	0	1	0	0	1	Rm			
Cond		0	1	I	P	U	B	W	L	Rn				Rd				Offest											
Cond		1	0	0	P	U	S	W	L	Rn				Register List															
Cond		0	0	0	P	U	1	W	L	Rn				Rd				Offset1		1	S	H	1	Offset2					
Cond		0	0	0	P	U	0	W	L	Rn				Rd				0	0	0	0	1	S	H	1	Rm			
Cond		1	0	1	L	Offset																							
Cond		0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn			
Cond		1	1	0	P	U	N	W	L	Rn				CRd				CPNum				Offset							
Cond		1	1	1	0	Op1				CRn				CRd				CPNum			Op2		0	CRm					
Cond		1	1	1	0	Op1			L	CRn				Rd				CPNum			Op2		1	CRm					
Cond		1	1	1	1	SWI Number																							

Conditional Execution

- Every instruction in ARM can be executed conditionally based on condition flag bits (previous ALU calculation)
- Significantly reduces branch costs causing pipeline stalls
- Condition fields
  - The condition fields of each instruction are compared with the condition flags in CPSR (Current Program Status Register)
  - The instruction can be executed only when the condition fields match condition flags in CPSR

Conditional Execution

- Condition to execute an instruction is described as postfix
  - Un-conditional execution

```
ADD r0,r1,r2          ; r0 = r1 + r2 (ADDAL)
```

  - Conditional execution

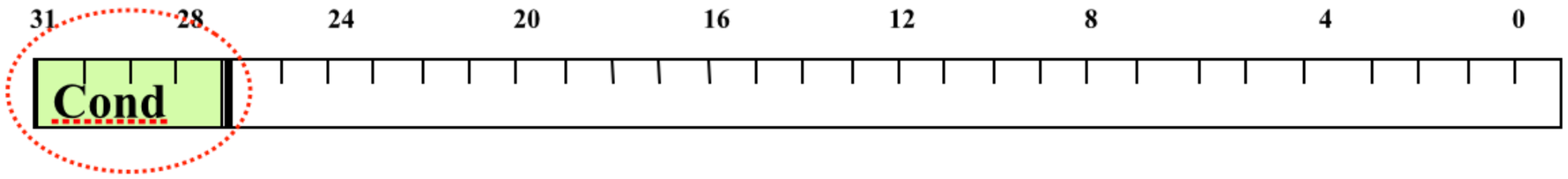
```
ADDEQ r0,r1,r2                ; If zero flag set then…
                              ; … r0 = r1 + r2
```

- Data processing instructions with postfix 'S' change condition flags in CPSR

```
SUBS r0,r1,r2                 ; r0 = r1 - r2
                              ; … and set flags
```

  - Comparison instructions always affect condition flags

Condition Field



Opcode [31:28]	Mnemonic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher or same	C set
0011	CC/LO	Carry clear / unsigned lower	C clear
0100	MI	Minus / negative	N set
0101	PL	Plus / positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N equals V
1011	LT	Signed less than	N is not equal to V
1100	GT	Signed greater than	Z clear and N equals V
1101	LE	Signed less than or equal	Z set or N is not equal to V
1110	AL	Always	any

ARM Instruction Set Overview

Mnemonics	ARM ISA	Description
ADC	v1	add two 32-bit values and carry
ADD	v1	add two 32-bit values
AND	v1	logical bitwise AND of two 32-bit values
B	v1	branch relative +/− 32 MB
BIC	v1	logical bit clear (AND NOT) of two 32-bit values
BKPT	v5	breakpoint instructions
BL	v1	relative branch with link
BLX	v5	branch with link and exchange
BX	v4T	branch with exchange
CDP CDP2	v2 v5	coprocessor data processing operation
CLZ	v5	count leading zeros
CMN	v1	compare negative two 32-bit values
CMP	v1	compare two 32-bit values
EOR	v1	logical exclusive OR of two 32-bit values
LDC LDC2	v2 v5	load to coprocessor single or multiple 32-bit values
LDM	v1	load multiple 32-bit words from memory to ARM registers
LDR	v1 v4 v5E	load a single value from a virtual address in memory
MCR MCR2 MCRR	v2 v5 v5E	move to coprocessor from an ARM register or registers
MLA	v2	multiply and accumulate 32-bit values
MOV	v1	move a 32-bit value into a register
MRC MRC2 MRRC	v2 v5 v5E	move to ARM register or registers from a coprocessor
MRS	v3	move to ARM register from a status register (*cpsr* or *spsr*)
MSR	v3	move to a status register (*cpsr* or *spsr*) from an ARM register
MUL	v2	multiply two 32-bit values
MVN	v1	move the logical NOT of 32-bit value into a register
ORR	v1	logical bitwise OR of two 32-bit values
PLD	v5E	preload hint instruction
QADD	v5E	signed saturated 32-bit add
QDADD	v5E	signed saturated double and 32-bit add
QDSUB	v5E	signed saturated double and 32-bit subtract
QSUB	v5E	signed saturated 32-bit subtract
RSB	v1	reverse subtract of two 32-bit values
RSC	v1	reverse subtract with carry of two 32-bit integers
SBC	v1	subtract with carry of two 32-bit values
SMLA*xy*	v5E	signed multiply accumulate instructions ($(16 \times 16) + 32 = 32$-bit)
SMLAL	v3M	signed multiply accumulate long ($(32 \times 32) + 64 = 64$-bit)
SMLAL*xy*	v5E	signed multiply accumulate long ($(16 \times 16) + 64 = 64$-bit)
SMLAW*y*	v5E	signed multiply accumulate instruction ($((32 \times 16) \gg 16) + 32 = 32$-bit)
SMULL	v3M	signed multiply long ($32 \times 32 = 64$-bit)