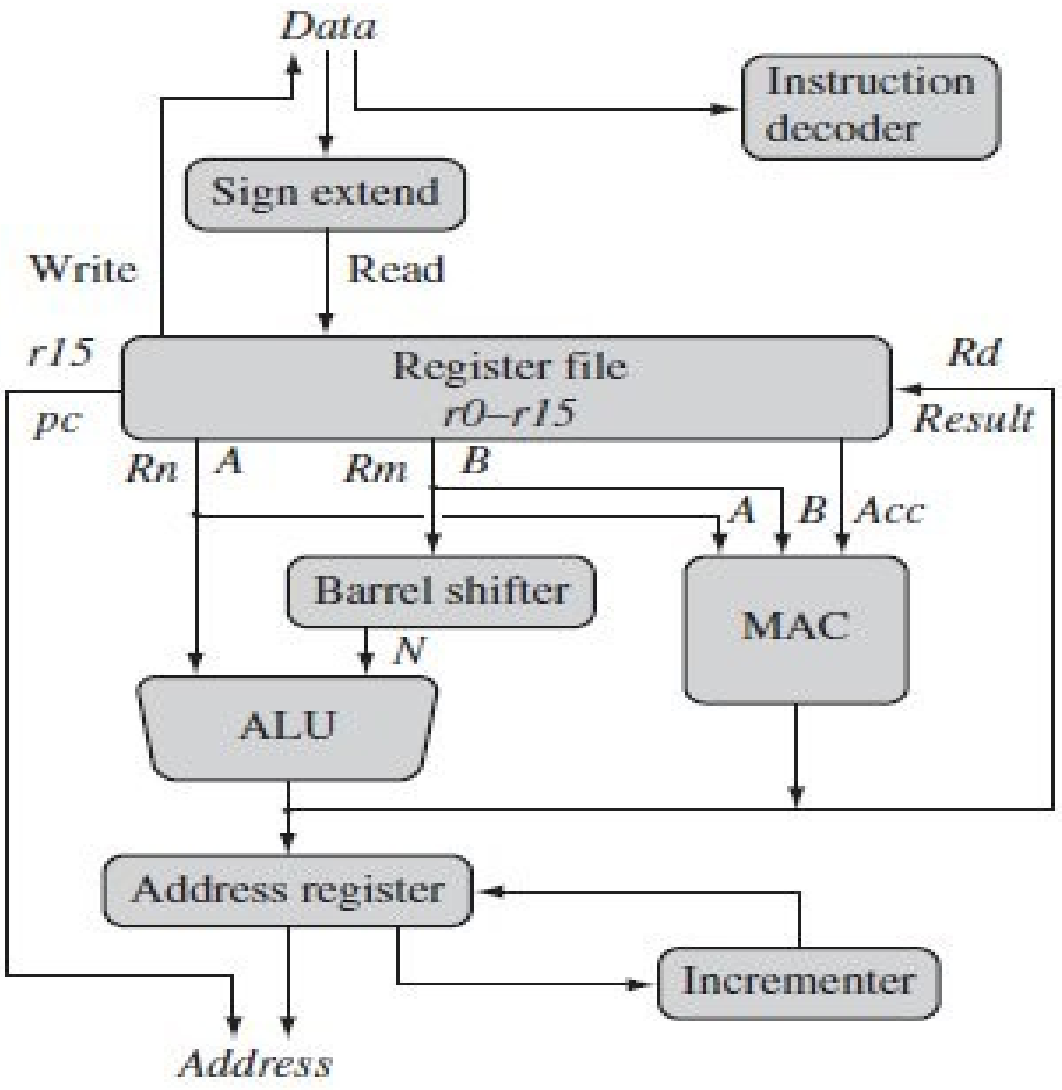
Embedded Systems

Lecture 2: ARM Processor fundamentals

ARM core dataflow model

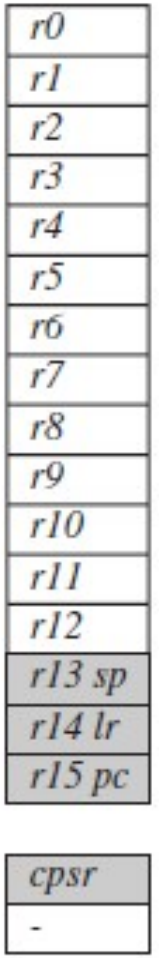


ARM Core data flow

- Von-Neumann implementation – data items and instructions share the same bus.
- Instruction decoder translates instructions before they are executed.
- Load instruction: copy data from memory to register
- Store instruction: copy data from register to memory
- There are no data processing instructions that directly manipulate data in memory.
- Data items are placed in the register file – a storage bank made up of 32-bit registers.
- ARM instructions typically have two source registers Rn and Rm and one destination register Rd.
- ALU and MAC (Multiply-accumulate) unit takes the register values Rn and Rm from A and B buses and computes a result.
- Load and store instructions use the ALU to generate an address to be held in the addres register and broadcast on the Address bus.
- The register Rm can be alternatively pre-processed in the barrel shifter before it enters the ALU.
- For load and store instructions the incrementer updates the address register before the core reads or writes the next register value from or to the next sequential memory location.

Registers

- General purpose registers hold either data or an address.
- All registers are 32-bit in size.
- 18 active registers
  - 16 data registers – r0 to r15
  - 2 process status registers
    - CPSR – Current Program Status Register
    - SPSR – Saved Program Status Register
- r13, r14, r15 have special functions
  - R13 – stack pointer (sp) and stores the head of the stack in the current processor mode
  - R14 – link register (lr) where the core puts the return address whenever it calls a subroutine.
  - R15 – programme counter (pc) and contains the address of the next instruction to be fetched by the processor.
- r13 and r14 can also be used as general purpose register as these registers are banked during a processor mode change.
- The registers r0 to r13 are orthogonal. Any instruction that you can apply to r0, you can equally apply to other registers.
- There are instructions that treat r14 and r15 in a special way.



Current Program Status Register (CPSR)

- ARM core uses CPSR to monitor and control internal operations.
- CPSR is a dedicated 32-bit register and resides in the register file.
- CPSR is divided into four fields each 8-bits wide:
  - Flags
  - Status
  - Extension - reserved for future use
  - Control – reserved for future use
- Control field contains processor modes, state and interrupt mask bits.

CPSR

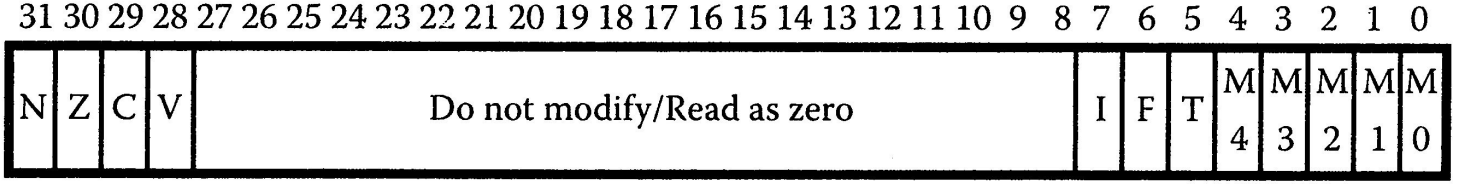
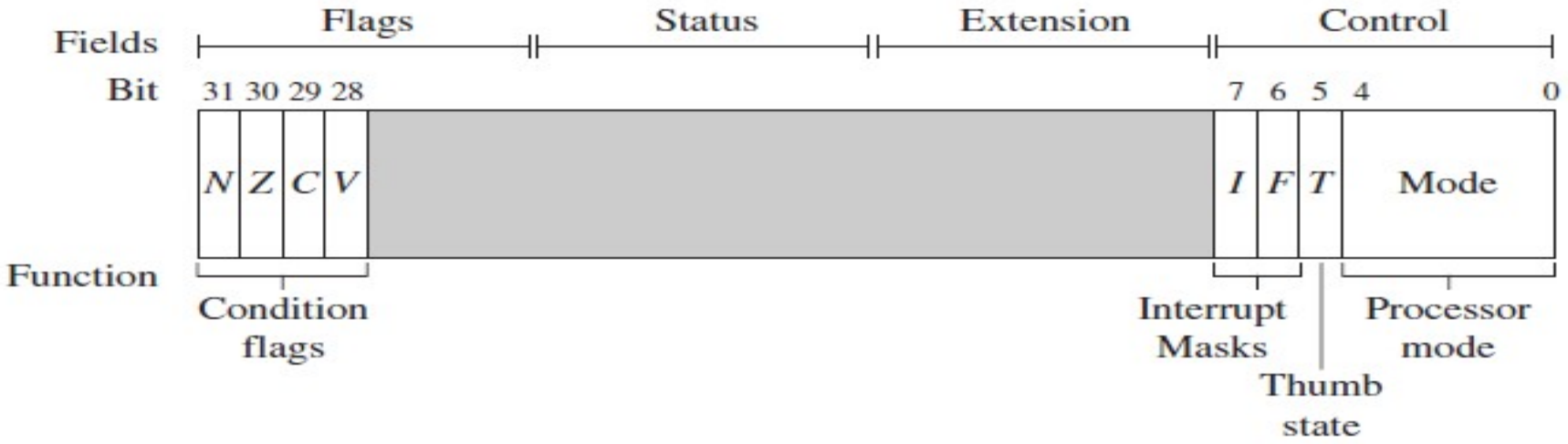


FIGURE 2.4 Format of the program status registers.



Processor Modes

- The processor mode determines which registers are active and the access rights to the CPSR register itself.
- Each processor mode is either privileged or non-privileged.
- A privileged mode allows full read-write access to CPSR.
- A non-privileged mode only allows read access to the control field in the CPSR but still allows R/W access to the condition flags.

Processor Modes

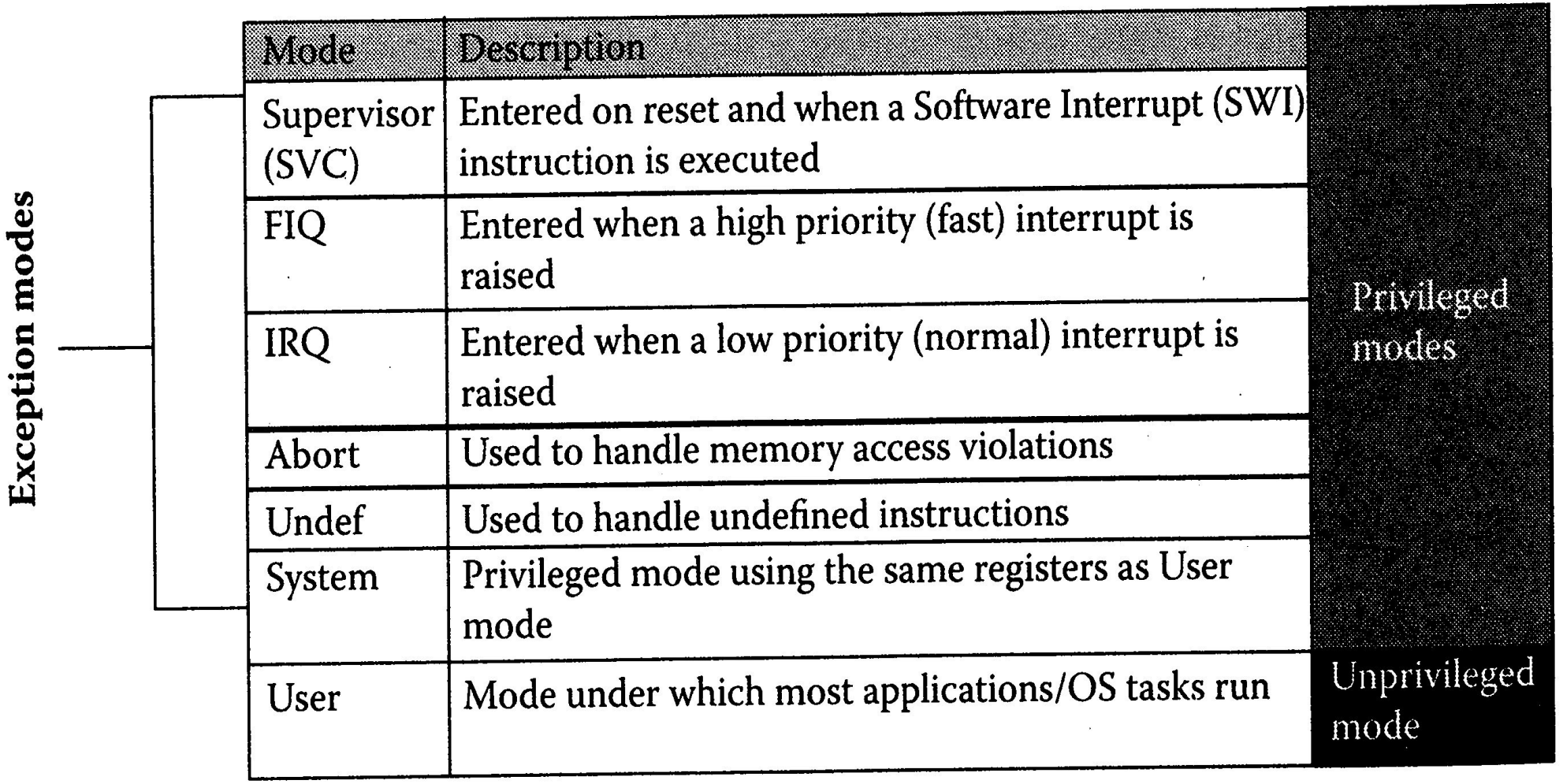
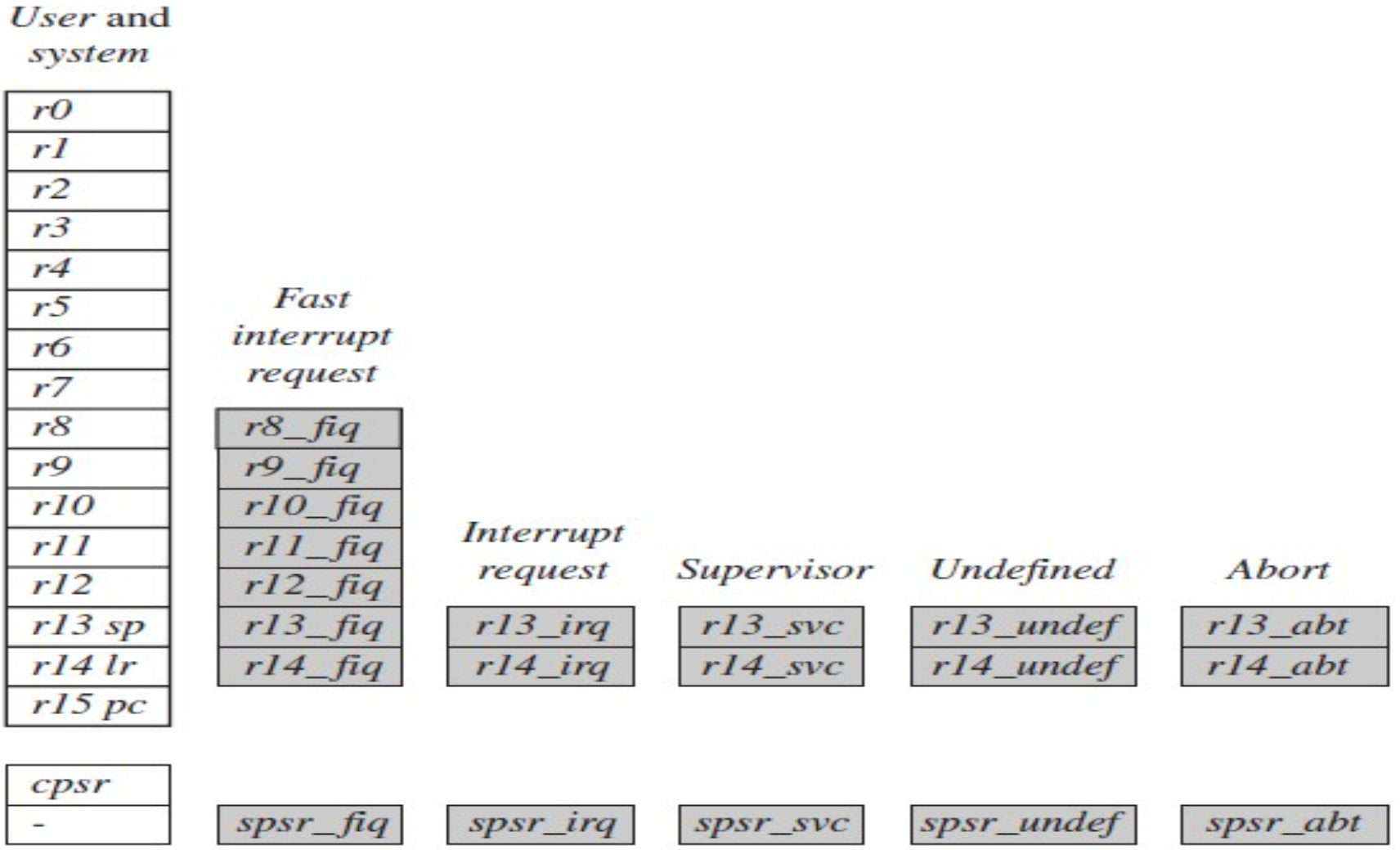


FIGURE 2.1 Processor modes.

Banked Registers

- ARM has 37 registers in the register file
- Of those, 20 registers are hidden from a program at different times.
- These registers are called banked registers (shown as shaded region)
- They are available only when the processor is in a particular mode.
- Every processor mode except user mode can change mode by writing directly to the mode bits of the CPSR.
- All modes except system mode have a set of associated banked registers that are a subset of the main 16 registers.
- A banked register maps one-to-one onto a user mode register.
- If you change processor mode, a banked register from the new mode will replace an existing register.
- There is no *spsr* available in the *user* mode.

Banked Registers Contd ...



Complete ARM register set.

Mode					
User/System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8_FIQ
R9	R9	R9	R9	R9	R9_FIQ
R10	R10	R10	R10	R10	R10_FIQ
R11	R11	R11	R11	R11	R11_FIQ
R12	R12	R12	R12	R12	R12_FIQ
R13	R13_SVC	R13_ABORT	R13_UNDEF	R13_IRQ	R13_FIQ
R14	R14_SVC	R14_ABORT	R14_UNDEF	R14_IRQ	R14_FIQ
PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_SVC	SPSR_ABORT	SPSR_UNDEF	SPSR_IRQ	SPSR_FIQ

= *banked register*

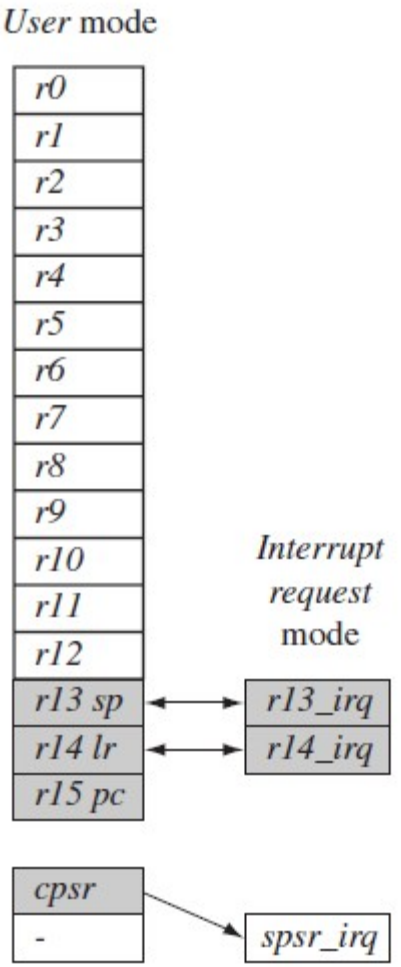
FIGURE 2.2 Register organization.

Changing Processor Mode

- The processor mode can be changed by a program that writes directly to CPSR (the processor has to be in privileged mode) or by hardware when core responds to an exception or interrupt.
- Following exceptions cause a mode change:
  - Reset
  - Interrupt Request
  - Fast interrupt request
  - Software interrupt
  - Data abort
  - Prefetch abort
  - Undefined instruction
- Exception and interrupt suspend the normal execution of sequential instructions and jump to a specific location.
-

Changing processor modes

- Core changing from user mode to interrupt request mode, which happens when an interrupt request occurs due to an external device raising an interrupt to the process core.
- This change causes user registers r13 and r14 to be banked.
- The user registers are replaced with registers *r13_irq* and *r14_irq* respectively.
- *r13_irq* contains the stack pointer for the interrupt request mode.
- *r14_irq* contains the return address.
- The *cpsr* is copied to *spsr_irq*.
- To return back to the user mode, a special return instruction is used that instructs the core to restore the original cpsr from spsr_irq and bank in the user registers r13 and r14.
- The register spsr can only be modified in a privileged mode. There is no spsr available in the user mode.
- Note that cpsr is not copied into the spsr when a mode change is forced due to a program writing directly to the cpsr. The saving of cpsr only occurs when an exception or interrupt is raised.



Changing mode on an exception.

Mode bits & Vector Table

Processor mode.

Mode	Abbreviation	Privileged	Mode[4:0]
Abort	abt	yes	10111
Fast interrupt request	fiq	yes	10001
Interrupt request	irq	yes	10010
Supervisor	svc	yes	10011
System	sys	yes	11111
Undefined	und	yes	11011
User	usr	no	10000

TABLE 2.2
Exception Vectors

Exception Type	Mode	Vector Address
Reset	SVC	0x00000000
Undefined instructions	UNDEF	0x00000004
Software Interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory abort)	ABORT	0x0000000C
Data abort (data access memory abort)	ABORT	0x00000010
IRQ (interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

Exception Vector Table

- The exception vector table consists of designated addresses in external memory that hold information to handle an exception, an interrupt or other atypical event (e.g. reset).
- For example, when an interrupt request comes along, the processor will change the program counter to 0x18 and begin fetching instructions from there.
- One can put a branch instruction at this location so that the processor can jump to the interrupt handler located at some other location in the memory.
- On reset, the core jumps to address 0x0 and starts fetching instruction. We either need to provide a reset exception handler (initialization routine) or we can begin coding at this address.

State & Instruction Sets

- The state of the core determines which instruction set is being executed.
- There are three instruction sets
  - ARM
  - Thumb
  - Jazelle
- The Jazelle (J) and Thumb (T) bits in the cpsr reflect the state of the processor.
- Jazelle instruction set is a closed set and is not openly available.

State & Instruction Sets Contd...

ARM and Thumb instruction set features.

	ARM (*cpsr* $T = 0$)	Thumb (*cpsr* $T = 1$)
Instruction size	32-bit	16-bit
Core instructions	58	30
Conditional execution[a]	most	only branch instructions
Data processing instructions	access to barrel shifter and ALU	separate barrel shifter and ALU instructions
Program status register	read-write in privileged mode	no direct access
Register usage	15 general-purpose registers +*pc*	8 general-purpose registers +7 high registers +*pc*

Jazelle instruction set features.

	Jazelle (*cpsr* $T = 0, J = 1$)
Instruction size	8-bit
Core instructions	Over 60% of the Java bytecodes are implemented in hardware; the rest of the codes are implemented in software.

Interrupt Masks & Condition Flags

- Interrupt masks are used to stop specific interrupt requests from interrupting the processor.
- Two types of interrupt request
  - Interrupt Request (IRQ)
  - Fast interrupt request (FIQ)
- Interrupt mask bits – I (7) and F (6)
- Condition flags are updated by comparison and the result of ALU operation that specify S instruction suffix. Eg. If a SUBS instruction results in a zero register value, Z flag in cpsr is set.
- Most ARM instructions can be executed conditionally on the value of the condition flags.
- Processors with DSP extensions, the Q bit indicates overflow or saturation that has occcured in an enhanced DSP instruction.
- Conditional execution controls whether or not the core will execute an instruction.
- Most instructions have a condition attribute which is compared with the condition flags in cpsr to determine if the instruction is to be executed.
- The condition attribute is post-fixed to the instruction mnemonic, which is encoded into the instruction.

Condition mnemonics.

Mnemonic	Name	Condition flags
`EQ`	equal	*Z*
`NE`	not equal	*z*
`CS HS`	carry set/unsigned higher or same	*C*
`CC LO`	carry clear/unsigned lower	*c*
`MI`	minus/negative	*N*
`PL`	plus/positive or zero	*n*
`VS`	overflow	*V*
`VC`	no overflow	*v*
`HI`	unsigned higher	*zC*
`LS`	unsigned lower or same	*Z* or *c*
`GE`	signed greater than or equal	*NV* or *nv*
`LT`	signed less than	*Nv* or *nV*
`GT`	signed greater than	*NzV* or *nzv*
`LE`	signed less than or equal	*Z* or *Nv* or *nV*
`AL`	always (unconditional)	ignored

Flag	Flag name	Set when
Q	Saturation	the result causes an overflow and/or saturation
V	oVerflow	the result causes a signed overflow
C	Carry	the result causes an unsigned carry
Z	Zero	the result is zero, frequently used to indicate equality
N	Negative	bit 31 of the result is a binary 1