Armv8-M Mainline Floating-point Extension

Learning Objectives

At the end of this topic, you will be able to:

- Summarize the Floating-point Extension features in the Armv8-M architecture
- Describe the Floating-point register bank and Floating-point system control registers
- Enable the Floating-point Extension from software
- Describe the mnemonic structure of Floating-point instructions
- Define the types of Floating-point exceptions

Agenda

- **Floating-point Extension overview**
- Registers
- Enabling the FPU
- Floating-point instructions
- Exceptions

Floating-point extension overview

- **The optional Floating-point Extension defines a Floating Point Unit (FPU)**
  - Only available in Mainline
- **Version of Floating-point Extension supported is FPv5**
- **IEEE 754-2008 compliant**
- **Provides:**
  - Single-precision arithmetic
  - Optional double-precision arithmetic (synthesis option)
  - 32 single-precision Floating-point registers S0-S31, also viewable as 16 double precision registers D0-D15
  - Data transfers between core registers and Floating-point registers of single-precision and double-precision values
  - Conversions between integer, half-precision, single-precision and double-precision formats
  - Software selectable Flush-to-zero mode

Floating-point extension overview

- **Encoding of instructions are for coprocessors 10 and 11**
  - Historical reasons only
  - Fully integrated into processor core

- **Adds system registers in coprocessor system space and System Control Block (SCB)**
  - System register in coprocessor system space – FPSCR
  - System registers in System Control Space – FPCAR, FPCCR, FPDSCR, MVFR0, MVFR1, MVFR2

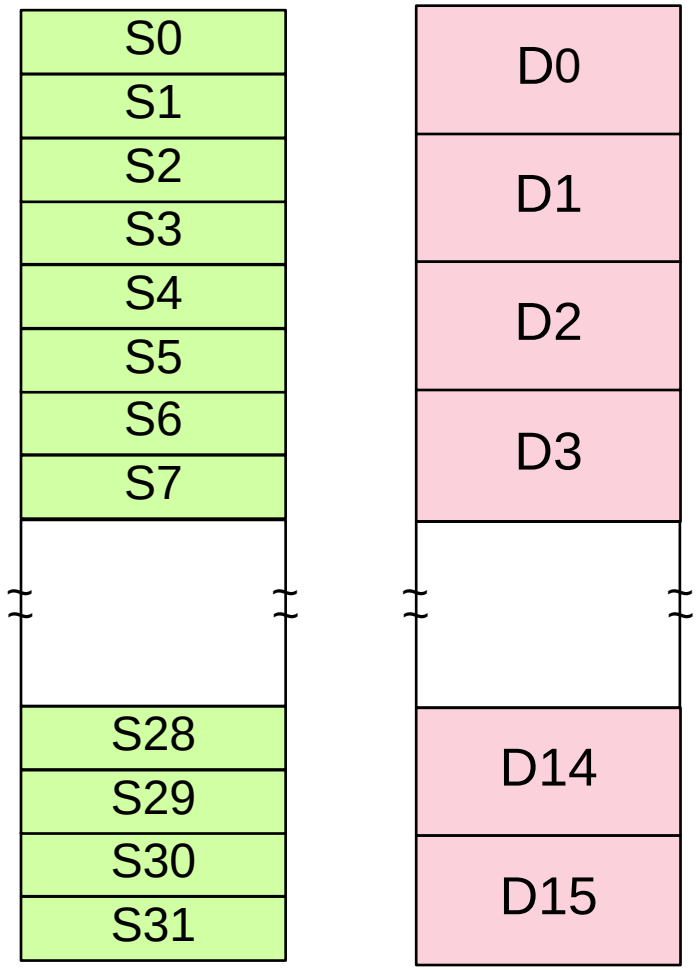
- **Software can interrogate Media and VFP Registers (MVFR0, MVFR1, MVFR2) to discover the implemented Floating-point features**

Agenda

- Floating-point Extension overview
- **Registers**
- Enabling the FPU
- Floating-point instructions
- Exceptions

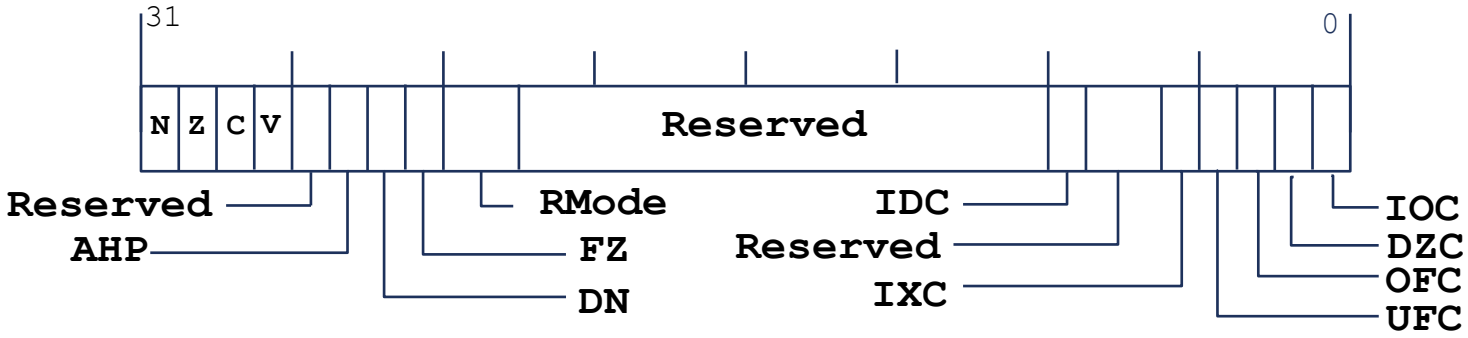
Data-processing Floating-point registers

- **FPU provides a 32 single-precision registers**
- **Can be viewed as either**
  - 32 x 32-bit registers
  - 16 x 64-bit doubleword registers
  - Any combination of the above
- **Not banked between security states**



Floating-point Status and Control Register

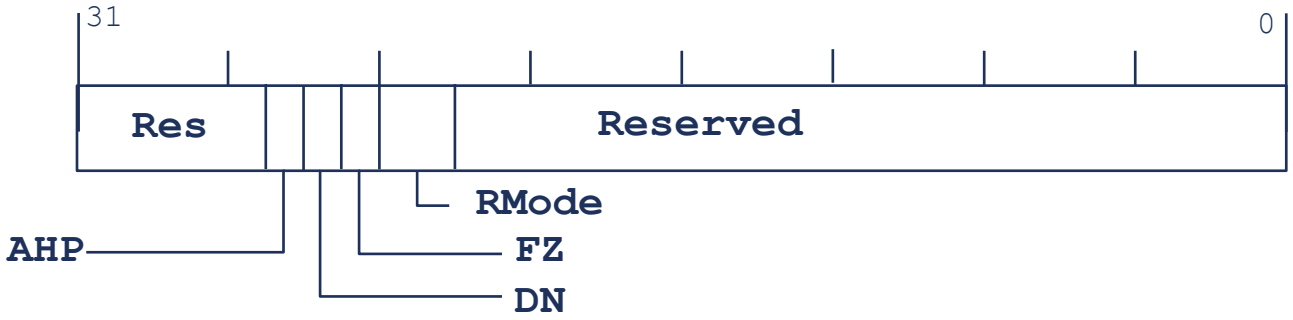
- **Floating-point Status and Control Register (FPSCR)**
- **Privileged access only, unprivileged access is RAZ/WI**
- **Not banked between security states**
- **Value is UNKNOWN on reset**



- **N, Z, C, V** – Floating-point comparison flags
- **AHP** – Alternative (non-IEEE 754-2008) half-precision
- **DN** – default NaN
- **FZ**- Flush-to-zero mode, any denormal inputs or results are flushed to zero (non-IEEE 754-2008)
- **Rounding Mode** Round to Nearest, Round towards Plus Infinity, Round towards Minus Infinity, Round towards Zero
- **IDC, IXC, IOC, DZC, OFC, UFC** – cumulative exception bits

Floating-point Default Status Control Register

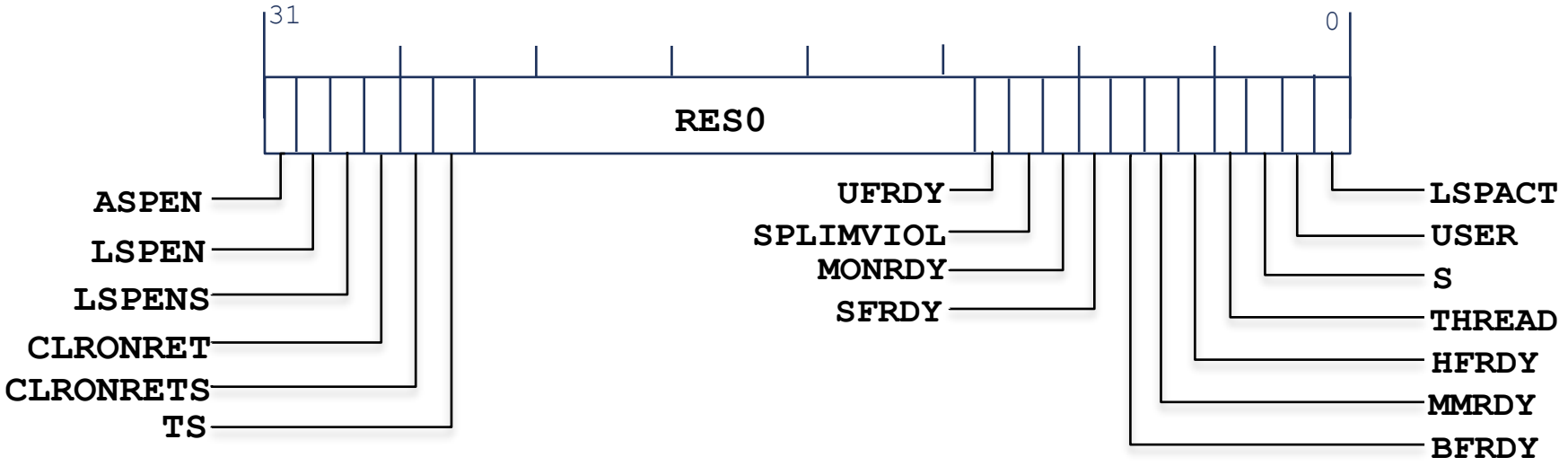
- Floating-point Default Status Control Register (FPDSCR)
- Holds default values for certain fields of the FPSCR when creating a new Floating-point context
- Privileged access only, unprivileged access generates a fault
- Banked between security states (non-secure version is FPDSCR_NS)
- All non-reserved fields set to zero on warm reset



- AHP – Alternative half-precision
- DN – Default NaN
- FZ – Flush-to-zero mode
- Rmode – Rounding mode

Floating-point Context Control Register

- Floating-point Context Control Register (FPCCR)
- Holds control data for the floating-point unit
- Privileged access only, unprivileged access generates a fault
- Banking between security states is on a per-bit basis
- Non-secure versions is FPCCR_NS, not all bits present in non-secure version



Agenda

- Floating-point Extension overview
- Registers
- **Enabling the FPU**
- Floating-point instructions
- Exceptions

Enabling the FPU in CPACR

- **Until the FPU is enabled, any Floating-point instruction or access to a register is `UNDEFINED`, causing a UsageFault exception**
- **To enable access to the FPU, the access bits [23:20] for CP10 and CP11 have to be enabled**
- **Coprocessor Access Control Register `CPACR` for secure state**
  - 0b00 == no access
  - 0b01 == privileged access only
  - 0b10 == **`RESERVED`**
  - 0b11 == any access
- **CP10 and CP11 access have to be set the same, otherwise result is `UNPREDICTABLE`**

```
LDR     R0, =0xE000ED88          ; Load address of CPACR
LDR     R1,[R0]                  ; Load current CPACR
ORR     R1,R1,#(0xF << 20)       ; Set CP10, CP11 any access
STR     R1,[R0]                  ; Write modified CPACR value
ISB                              ; Instruction Synchronization Barrier
```

Enabling the FPU for Non-secure state

- **Non-Secure Access Control Register should be programmed to enable FPU in Non-secure state along with CPACR settings**

- **Non-secure Access Control Register `NSACR` for non-secure state**
  - 0b0 == non-secure accesses generate NOCP fault
  - 0b1 == any access is allowed

- **CP10 and CP11 access have to be set the same, otherwise result is `UNPREDICTABLE`**

```
LDR     R0, =0xE000ED8C                 ; Load address of NSACR
LDR     R1,[R0]                         ; Load current NSACR
ORR     R1,R1,#(0x3 << 10)              ; Set CP10, CP11 non-secure access allowed
STR     R1,[R0]                         ; Write modified NSACR value
ISB                                     ; Instruction synchronization barrier
```

Enabling the FPU in CPACR (via CMSIS)

https://github.com/ARM-software/CMSIS_5/blob/develop/Device/ARM/ARMCM33/Source/system_ARMCM33.c

```
76  void SystemInit (void)
77  {
78
79  #if defined (__VTOR_PRESENT) && (__VTOR_PRESENT == 1U)
80    SCB->VTOR = (uint32_t) &(__VECTOR_TABLE[0]);
81  #endif
82
83  #if defined (__FPU_USED) && (__FPU_USED == 1U)
84    SCB->CPACR |= ((3U << 10U*2U) |           /* enable CP10 Full Access */
85                   (3U << 11U*2U)  );         /* enable CP11 Full Access */
86  #endif
87
88  #ifdef UNALIGNED_SUPPORT_DISABLE
89    SCB->CCR |= SCB_CCR_UNALIGN_TRP_Msk;
90  #endif
```

```
LDR     R0, =0xE000ED88
LDR     R1,[R0]
ORR     R1,R1,#(0xF << 20)
STR     R1,[R0]
ISB
```

Agenda

- Floating-point Extension overview
- Registers
- Enabling the FPU
- **Floating-point instructions**
- Exceptions

Floating-point instructions (1)

- **Floating-point instructions are of the form:**

  `Vop{<c>}{<q>}{.<dt>} {<dest>,} <src1>, <src2>`

- **Where:**
  - <c> and <q> are standard assembler condition and encoding qualifier syntax fields
  - <dt> is a data type specifier, of the form:
    - Floating-point - F16 (half), F32 (single), or F64 (double). F32 can be replaced with F, F64 with D
    - More than one <dt> specifier can be applied to an instruction
- **All Floating-point operations start with a V prefix**
- **F64 only supported for register loads and stores on single-precision variants**
- **F16 only supported for conversion instructions**

Floating-point instructions (2)

- **`VMOV` used to transfer data between core (`Rn`) and Floating-point (`Sn`/`Dn`) registers:**
  - From one R register to one S register
  - From two R registers to two S registers
  - From one S register to one R register
  - From two S registers to two R registers
  - From two R registers to one D register
- **`VMOV` immediate available**
- **`VMRS` and `VMSR` is used to move a core (`Rn`) register to and from `FPSCR`**
- **Transfers of values to and from memory, usual instructions, just with `V` prefix**
  - `VLDR, VLDM, VSTR, VSTM, VPUSH, VPOP`
- **Lazy load multiple and lazy store multiple : `VLLDM`, `VLSTM`**
  - only available when Security Extension is implemented

Floating-point instructions (3)

- Standard arithmetic instructions: `VADD, VSUB, VMUL, VDIV`
- Fused multiply accumulate, fused multiply subtract: `VMLA, VMLS`
- Multiply and negate: `VNMUL`
- Fused negate multiply accumulate, fused negate multiply subtract - negates the multiply before addition/subtraction: `VNMLA, VNMLS`
- Negate: `VNEG`
- Select minimum or maximum: `VMIN, VMAX`
- Square root (based on method of least squares): `VSQRT`
- Half-precision conversions: `VCVTB` (bottom) `VCVTT` (top)
- Floating-point compares: `VCMP, VCMPE`
- Conditional selection selects from one of two registers depending on `FPSCR` condition codes: `VSEL`

Agenda

- Floating-point Extension overview
- Registers
- Enabling the FPU
- Floating-point instructions
- **Exceptions**

Floating-point exceptions

- **Floating-point exceptions are not hardware exceptions**
- **Five IEEE 754-2008 defined Floating-point exceptions**
  - Inexact (`IXC`)
  - Underflow (`UFC`)
  - Overflow (`OFC`)
  - Division by Zero (`DZC`)
  - Invalid Operation (`IOC`)
- **One Arm-specific Floating-point exception**
  - Input Denormal (IDC)
  - Can only be set in Flush-to-zero mode (`FPSCR.FZ` set to 1)
  - Half-precision numbers are exempt from Flush-to-zero mode
- **Once set, stay set until cleared by software (the ‘C’ stands for ‘Cumulative’)**

Exception handling

- **Four possible modes for stacking FP context information on taking an exception:**

- **Do not stack any FP context**
  - The processor stacks only a Basic frame
- **Stack an Extended frame**
  - Containing the Basic frame and the FP state information
  - This preserves the Floating-point state required by the AAPCS
- **Reserve space on the stack for an Extended frame, but write only the Basic frame information**
  - No data is pushed into the stack locations reserved for `S0-S15` and the `FPSCR` value
  - This is an FP lazy context save
- **Stack entire Floating-point context**
  - All of `S0-S31` and the `FPSCR` is stacked

- **The `ASPEN`, `LSPEN` and `TS` bits in `FPSCR`, and `CONTROL.FPCA` determine which action is taken, along with the security state at the time of the exception, and the security of the exception being taken …**

Floating-point context control

- **Bit 2 of `CONTROL` register, `FPCA`, indicates Floating-point Extension is active**
  - Reset value is 0
  - Set automatically using FPCCR.ASPEN (see below)
  - When set, hardware preserves Floating-point context on exception entry, if necessary, and restores on exception return
- **On exception entry, `EXC_RETURN[4]` is set to inverse of `CONTROL.FPCA`**
  - `EXC_RETURN` value of `0xFFFFFFEx` indicates extended stack frame
- **Floating-point Context Control Register (FPCCR)**
  - Bit 31 `ASPEN` – when set (default), successful execution of any Floating-point instruction sets `CONTROL.FPCA` to 1
  - Bit 30 `LSPEN` – when set (default), enables Extended stack frame when set
  - Only accessible by privileged software

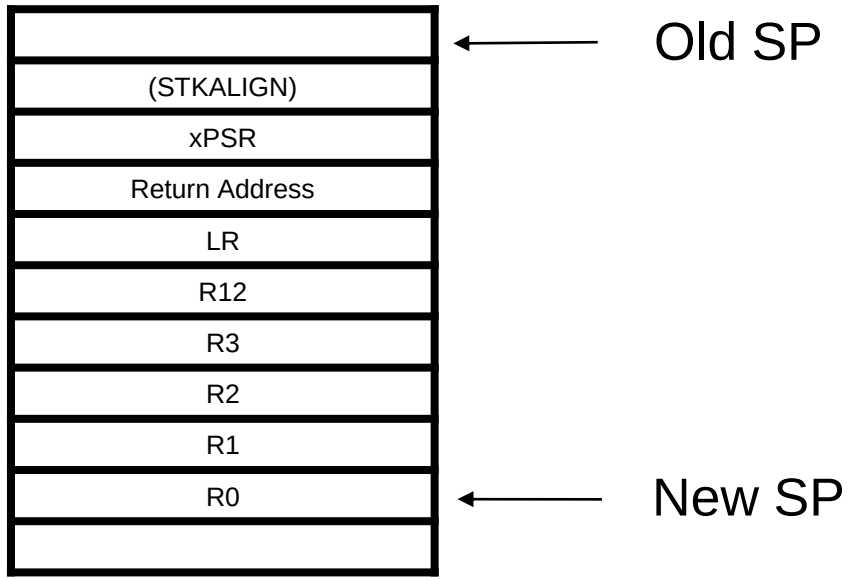
ASPEN=0 LSPEN=0	ASPEN=0 LSPEN=1	ASPEN=1 LSPEN=0	ASPEN=1 LSPEN=1
• Stack Basic frame	• Invalid configuration	• Automatic FP stacking • Stack Extended frame	• Reserve space for Extended stack fame • Stack Basic frame • Set LPPACT to 1 • Stack FP registers only if used in interrupt handler

ASPEN
Automatic State Preservation Enable

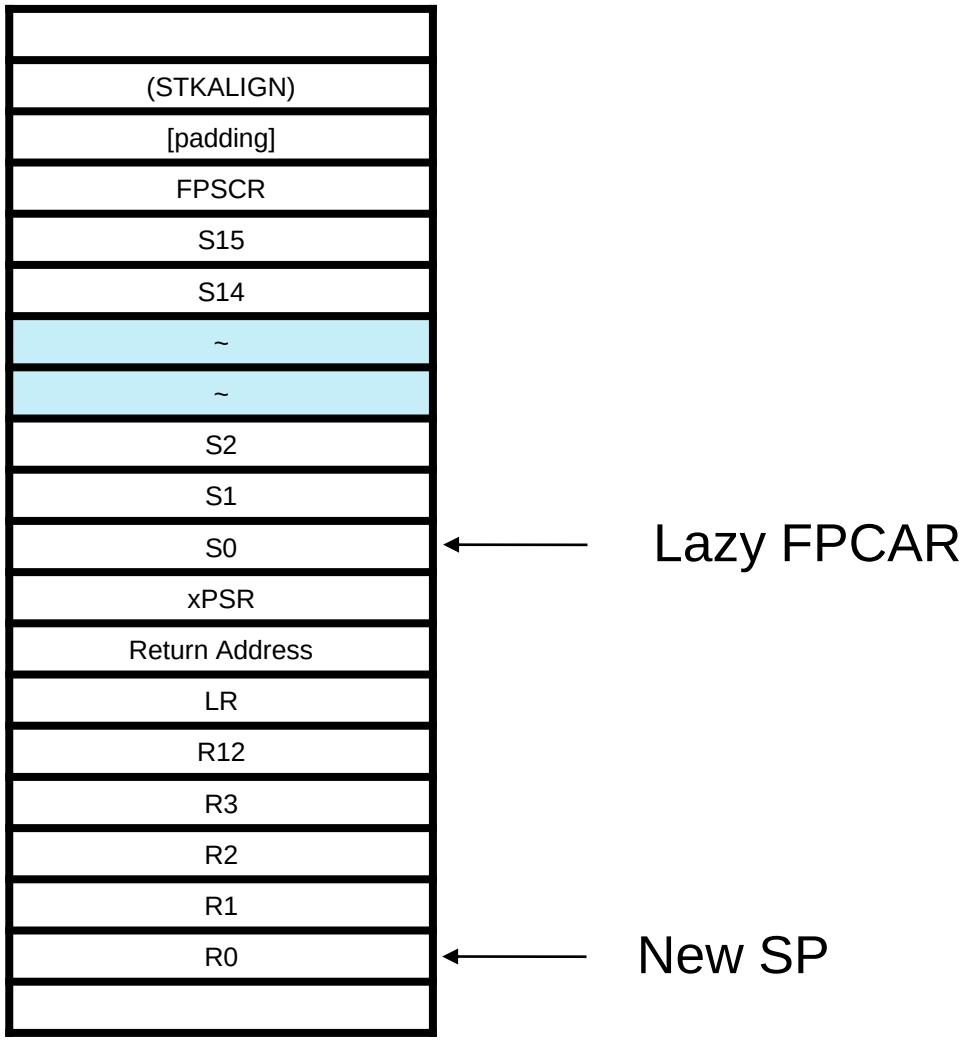
LSPEN
Lazy State Preservation Enable

LPPACT
Lazy State Preservation Active

Basic versus Extended frame



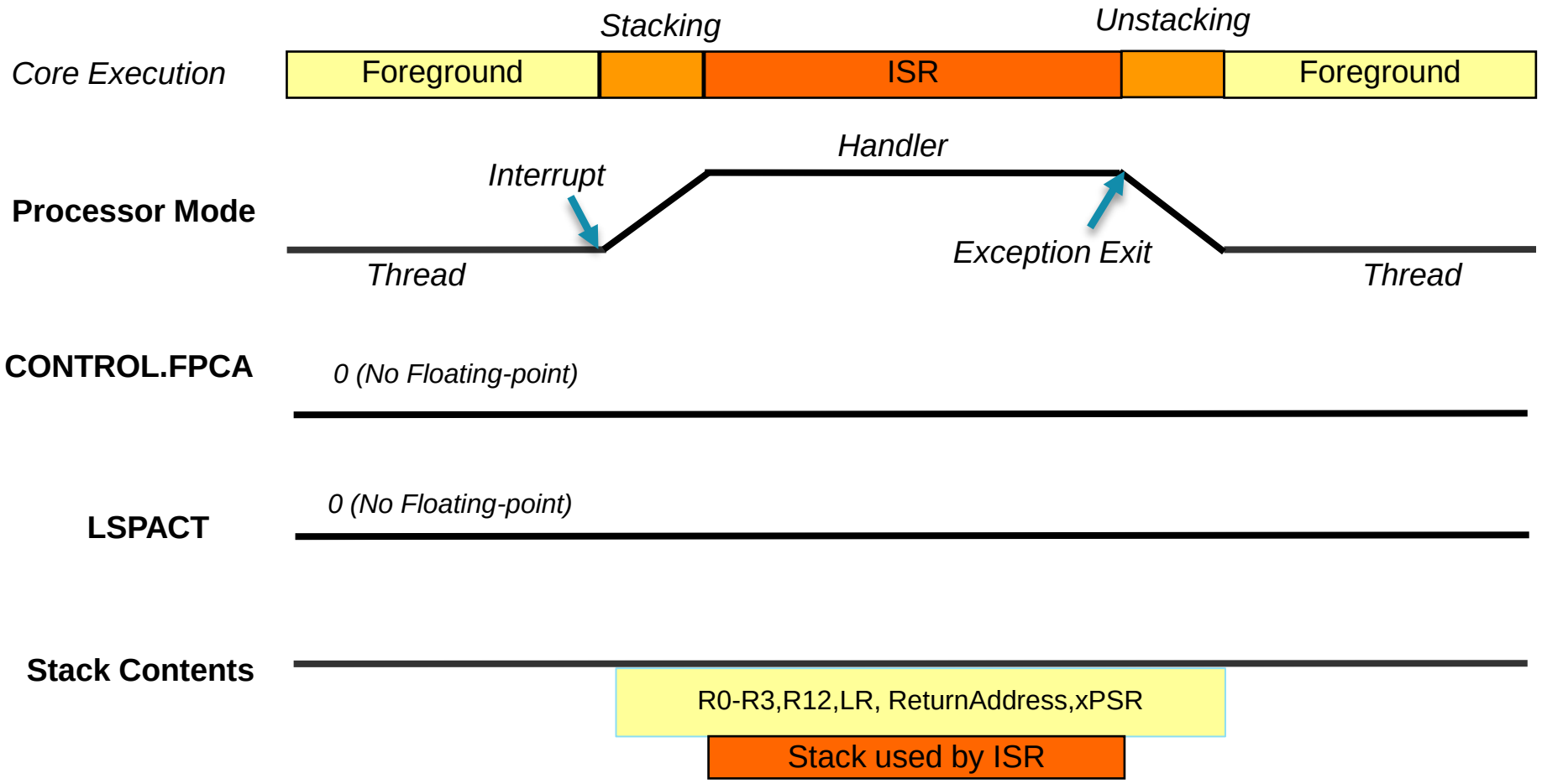
Basic stack frame, no FP context



Extended stack frame with FP context

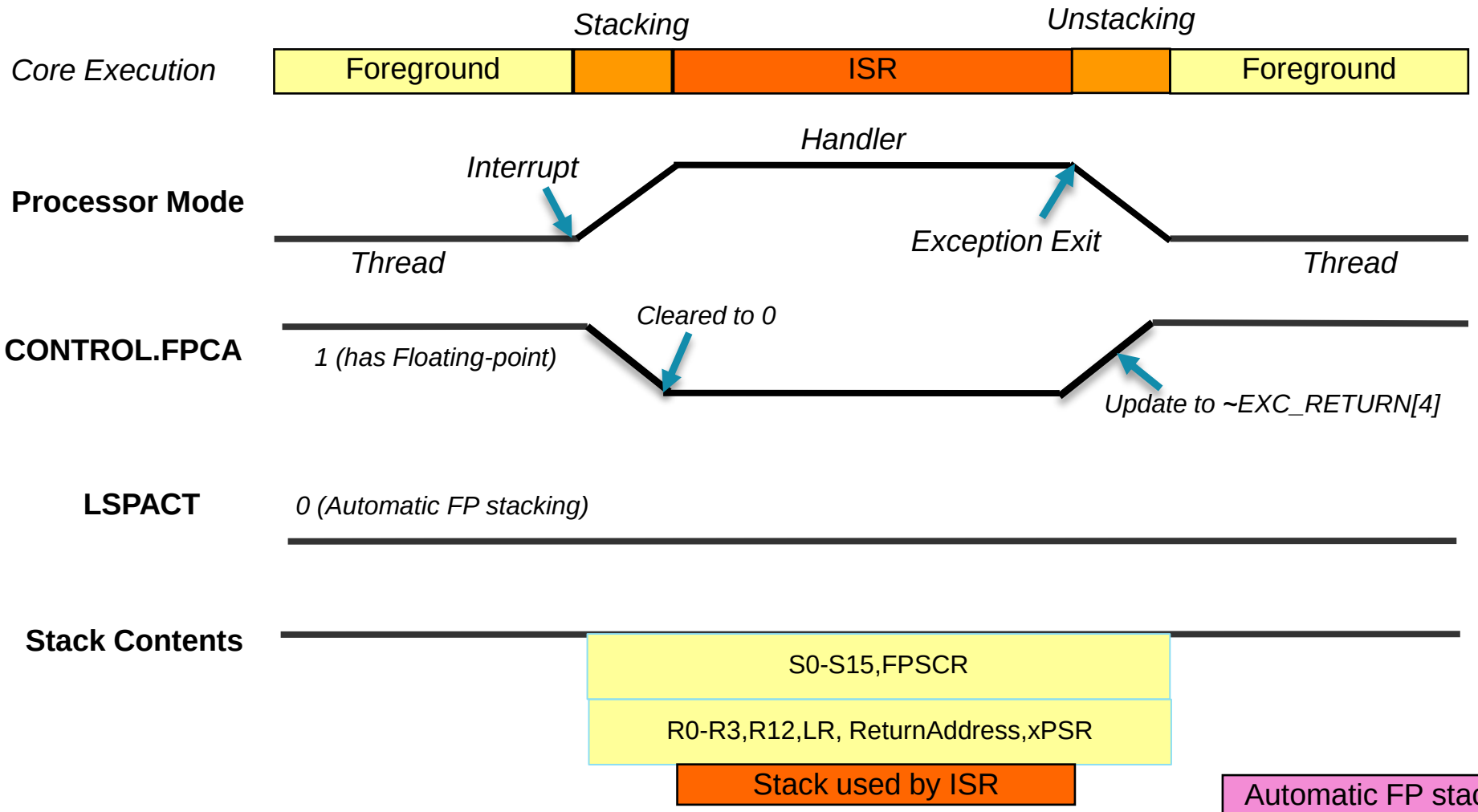
- **One word of padding always included in extended stack frame**
- **Optional word of padding added if required for 8-byte stack alignment**

Example 1 – No Floating-point context



Basic Integer stacking when FPCCR.ASPEN = 0 and FPCCR.LSPEN = 0

Example:2 With Floating-point context (Automatic)



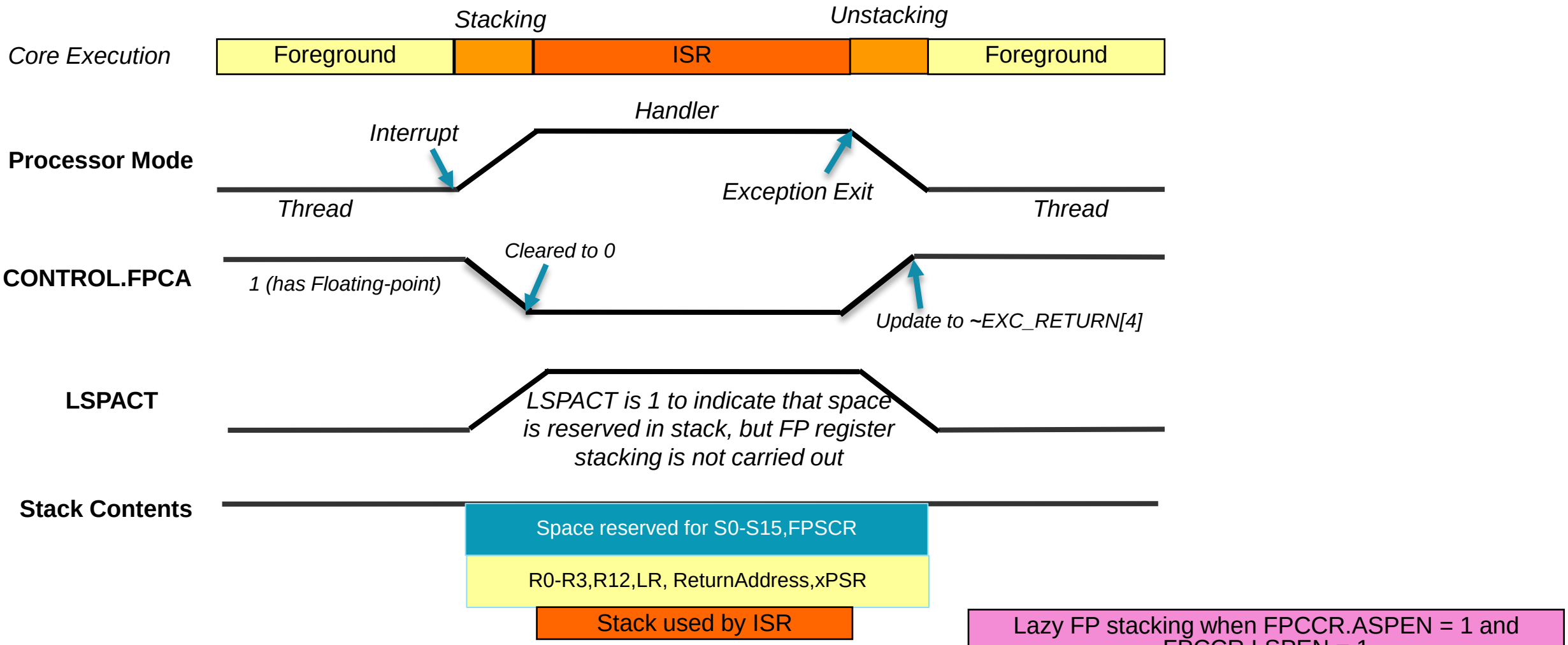
Automatic FP stacking when FPCCR.ASPEN = 1 and FPCCR.LSPEN = 0

Extended Stack Frame with Floating-point registers

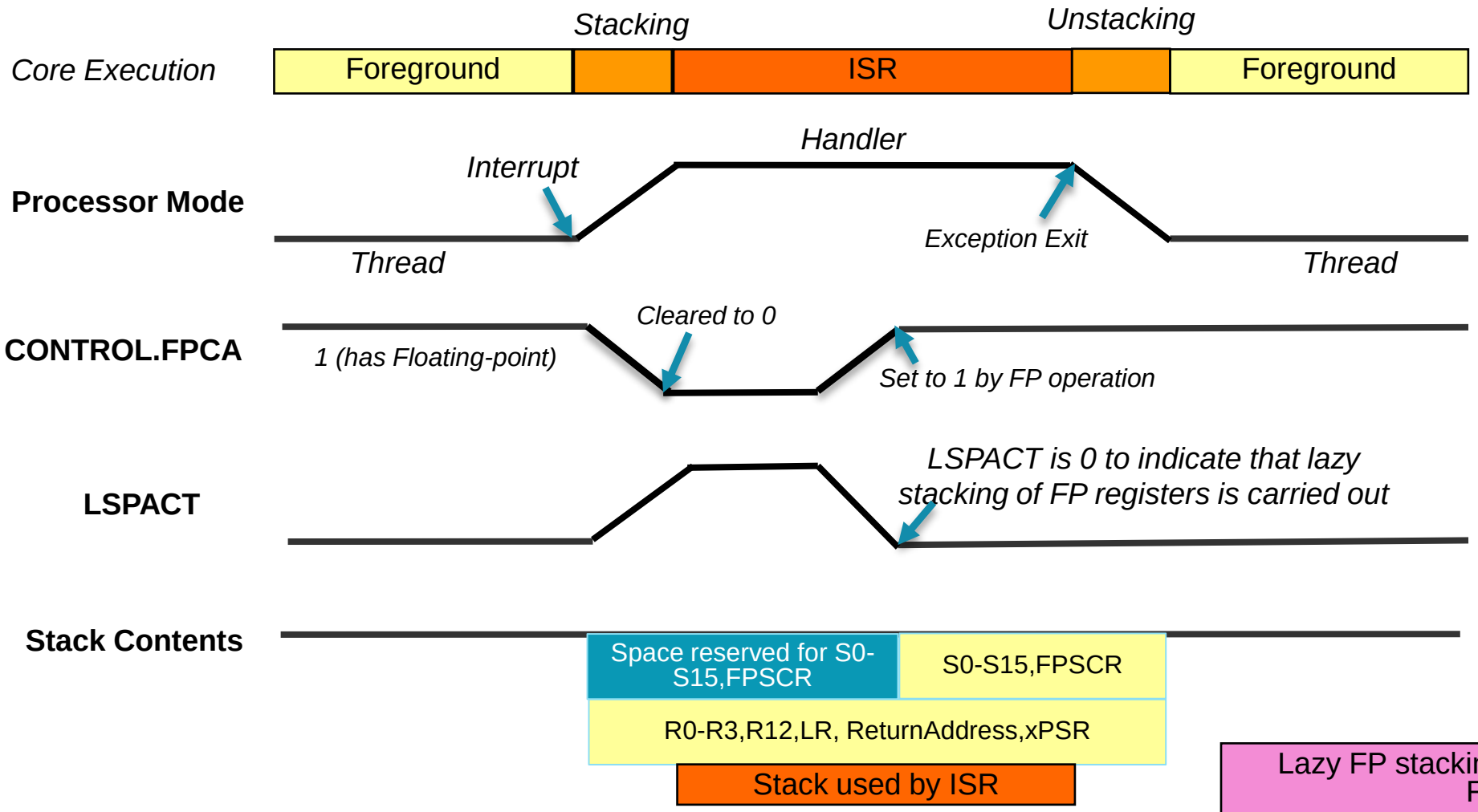
Lazy context save

- **Pushing 17 words onto the stack is expensive**
- **Exception handlers or tasks do not have to use Floating-point registers**
- **If `FPCCR.LSPEN` is set, then on exception, the stack pointer is incremented as if the Floating-point context is saved, but the push is not performed**
- **Exception handler can then choose whether to stack or not**
- **Floating-point Context Address Register (FPCAR) holds the address location of the unpopulated but allocated space in the FP stack frame when lazy stacking is in use**
  - Banked between security state
  - Non-secure version `FPCAR_NS`, if Security Extension present
- **IMPORTANT – `FPCCR.LSPEN` cannot be set while `CONTROL.FPCA` is set**
  - Therefore, lazy context switch must be enabled before user application Floating-point code starts executing

Example:3 – Floating-point context in Foreground (Lazy)



Example:4 – Floating-point context in ISR (Lazy)



Lazy FP stacking when FPCCR.ASPEN = 1 and FPCCR.LSPEN = 1

Lazy stacking carried out with space reserved in Stack

Interaction with the Security Extension

- **When the PE also contains the Security Extension, there is an additional interaction**

- **If the the PE is in the the secure state, there is a Floating-point context active, and a non-secure exception is taken:**
  - All the general purpose registers are stacked
  - All 32 Floating-point registers are stacked
  - The xPSR and FPSCR are stacked
  - There are two alignment words which are reserved on the stack
  - There is an integrity signature stacked
  - The return address is stacked

- **This makes the stack frame 51 words long, which is extremely expensive**

- **Think very carefully about using Floating-point in the secure state**

References

- **Armv8-M Architecture Reference Manual**
- **Application Note 298 - Cortex-M4(F) Lazy Stacking and Context Switching**
- **“The Definitive guide to Arm Cortex-M23 and Cortex-M33 processors” – By Joseph Yiu**

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה