Technische Hochschule Rosenheim
Technical University of Applied Sciences

Prof. Dr. Florian Künzner
Technical University of Applied Sciences Rosenheim, Computer Science

CA 6 – Processor 3

The lecture is based on the work and the documents of Prof. Dr. Theodor Tempelmeier

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goal

CAMPUS Rosenheim
Computer Science

Goal

CA::Processor 3

- Registers
- Processor examples
- Instructions
- Addressing modes

CAMPUS Rosenheim
Computer Science

Registers overview

Register properties

- Registers are very fast
- Configuration and start-up of certain features
- Status reporting such as whether a certain event has occurred
- Registers can be read/write or read-only
- Ordered as register file (array of registers)

Register usage

- Registers for data
- Registers for addresses (PC, SP)
- Registers for status (SR)
- Registers for controlling CPU modes

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Registers overview

Register properties

- Registers are very fast
- Configuration and start-up of certain features
- Status reporting such as whether a certain event has occurred
- Registers can be read/write or read-only
- Ordered as register file (array of registers)

Register usage

- Registers for data
- Registers for addresses (PC, SP)
- Registers for status (SR)
- Registers for controlling CPU modes

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Registers overview

Register properties

- Registers are very fast
- Configuration and start-up of certain features
- Status reporting such as whether a certain event has occurred
- Registers can be read/write or read-only
- Ordered as register file (array of registers)

Register usage

- Registers for data
- Registers for addresses (PC, SP)
- Registers for status (SR)
- Registers for controlling CPU modes

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Registers overview

Register properties

- Registers are very fast
- Configuration and start-up of certain features
- Status reporting such as whether a certain event has occurred
- Registers can be read/write or read-only
- Ordered as register file (array of registers)

Register usage

- Registers for data
- Registers for addresses (PC, SP)
- Registers for status (SR)
- Registers for controlling CPU modes

CAMPUS Rosenheim
Computer Science

Registers overview

Register properties

- Registers are very fast
- Configuration and start-up of certain features
- Status reporting such as whether a certain event has occurred
- Registers can be read/write or read-only
- Ordered as register file (array of registers)

Register usage

- Registers for data
- Registers for addresses (PC, SP)
- Registers for status (SR)
- Registers for controlling CPU modes

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Registers overview

Register properties

- Registers are very fast
- Configuration and start-up of certain features
- Status reporting such as whether a certain event has occurred
- Registers can be read/write or read-only
- Ordered as register file (array of registers)

Register usage

- Registers for data
- Registers for addresses (PC, SP)
- Registers for status (SR)
- Registers for controlling CPU modes

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Registers overview

Register properties

- Registers are very fast
- Configuration and start-up of certain features
- Status reporting such as whether a certain event has occurred
- Registers can be read/write or read-only
- Ordered as register file (array of registers)

Register usage

- Registers for data
- Registers for addresses (PC, SP)
- Registers for status (SR)
- Registers for controlling CPU modes

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Registers overview

Register properties

- Registers are very fast
- Configuration and start-up of certain features
- Status reporting such as whether a certain event has occurred
- Registers can be read/write or read-only
- Ordered as register file (array of registers)

Register usage

- Registers for data
- Registers for addresses (PC, SP)
- Registers for status (SR)
- Registers for controlling CPU modes

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Registers overview

Register properties

- Registers are very fast
- Configuration and start-up of certain features
- Status reporting such as whether a certain event has occurred
- Registers can be read/write or read-only
- Ordered as register file (array of registers)

Register usage

- Registers for data
- Registers for addresses (PC, SP)
- Registers for status (SR)
- Registers for controlling CPU modes

Program Counter (IP - Instruction Pointer)
Stack Pointer

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Registers overview

Register properties

- Registers are very fast
- Configuration and start-up of certain features
- Status reporting such as whether a certain event has occurred
- Registers can be read/write or read-only
- Ordered as register file (array of registers)

Register usage

- Registers for data
- Registers for addresses (PC, SP)
- Registers for status (SR) - Status Register
- Registers for controlling CPU modes

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Registers overview

Register properties

- Registers are very fast
- Configuration and start-up of certain features
- Status reporting such as whether a certain event has occurred
- Registers can be read/write or read-only
- Ordered as register file (array of registers)

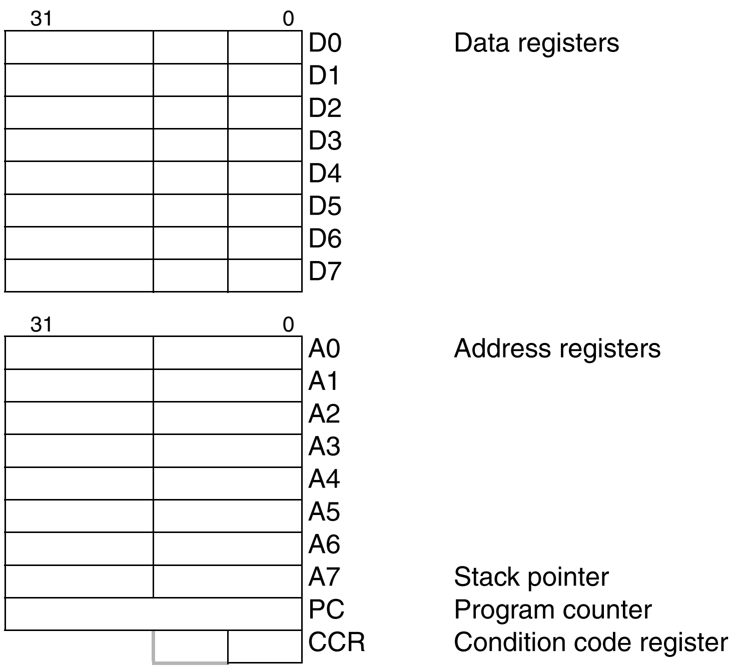
Register usage

- Registers for data
- Registers for addresses (PC, SP)
- Registers for status (SR)
- Registers for controlling CPU modes

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example: Freescale ColdFire (1)



Freescale ColdFire

- 32 bit architecture
- 16 general purpose 32 bit registers (D0-D7, A0-A7)
- 8 bit condition code register/16 bit status register

Address registers

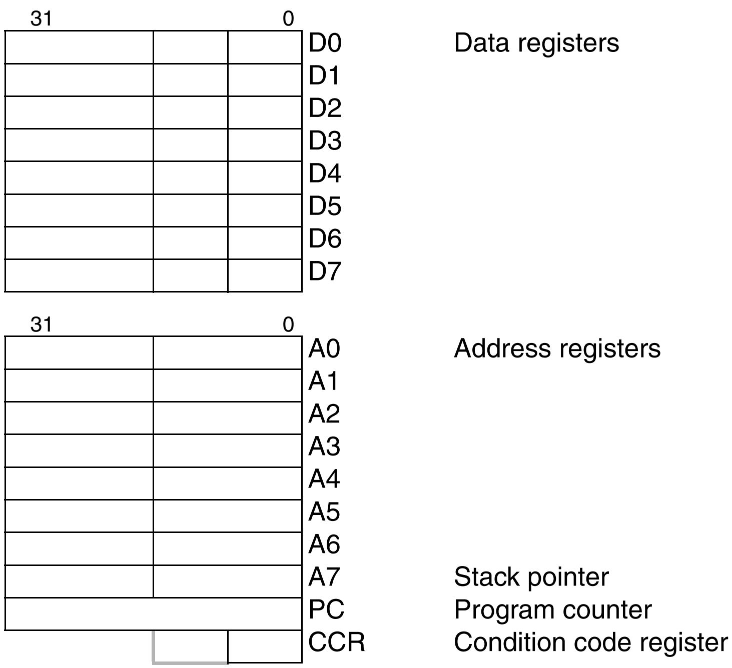
- Addresses of variables
- Base address registers
- Index address registers
- Stack pointer (SP)
- Program counter (PC)
  Address of currently executed instruction

[source: [ColdFire Family Programmers Reference]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example: Freescale ColdFire (1)



[source: [ColdFire Family Programmers Reference]]

Freescale ColdFire

- 32 bit architecture
- 16 general purpose 32 bit registers (D0-D7, A0-A7)
- 8 bit condition code register/16 bit status register

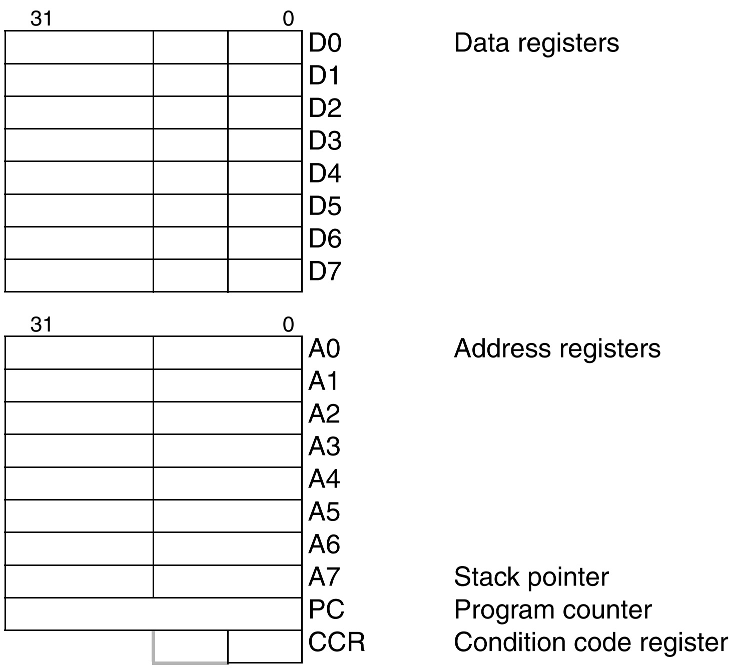
Address registers

- Addresses of variables
- Base address registers
- Index address registers
- Stack pointer (SP)
- Program counter (PC)
  Address of currently executed instruction

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example: Freescale ColdFire (1)



Freescale ColdFire

- 32 bit architecture
- 16 general purpose 32 bit registers (D0-D7, A0-A7)
- 8 bit condition code register/16 bit status register

Address registers

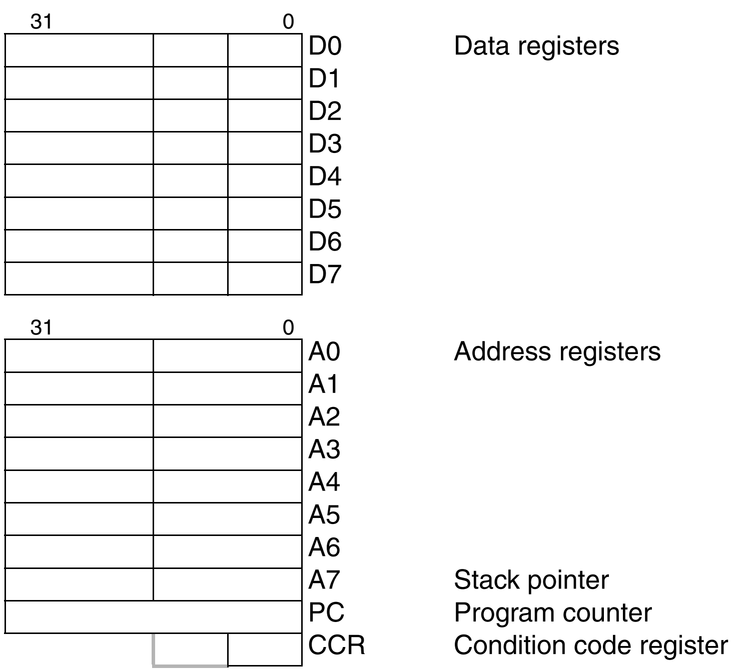
- Addresses of variables
- Base address registers
- Index address registers
- Stack pointer (SP)
- Program counter (PC)
  Address of currently executed instruction

[source: [ColdFire Family Programmers Reference]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example: Freescale ColdFire (1)



[source: [ColdFire Family Programmers Reference]]

Freescale ColdFire

- 32 bit architecture
- 16 general purpose 32 bit registers (D0-D7, A0-A7)
- 8 bit condition code register/16 bit status register

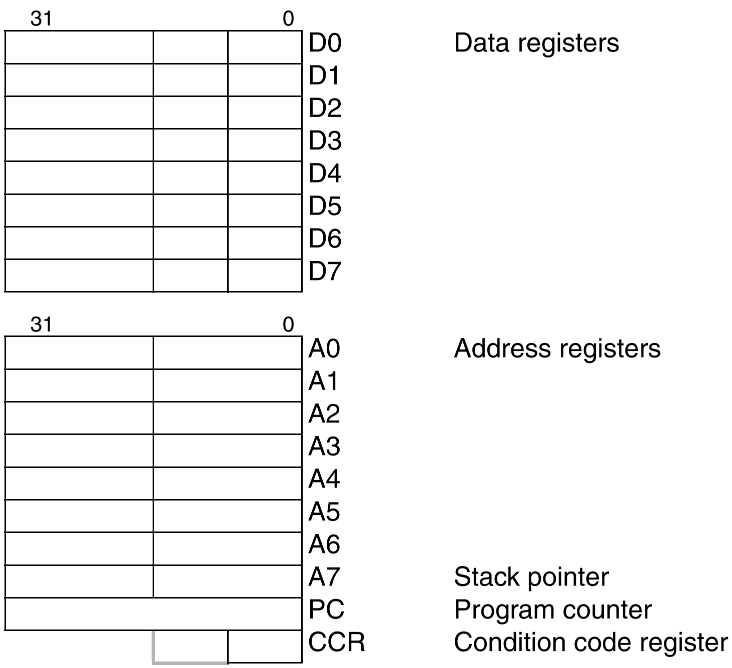
Address registers

- Addresses of variables
- Base address registers
- Index address registers
- Stack pointer (SP)
- Program counter (PC)
  Address of currently executed instruction

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example: Freescale ColdFire (1)



Freescale ColdFire

- 32 bit architecture
- 16 general purpose 32 bit registers (D0-D7, A0-A7)
- 8 bit condition code register/16 bit status register

Address registers

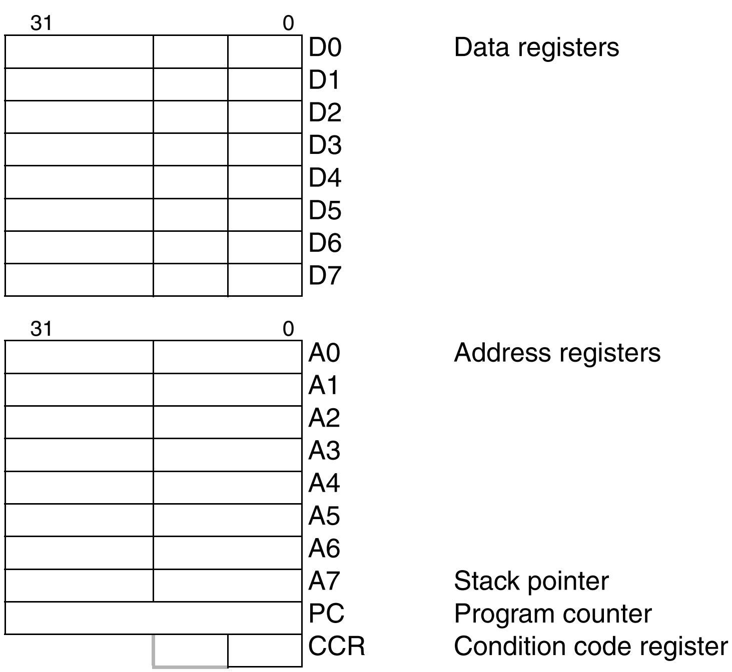
- Addresses of variables
- Base address registers
- Index address registers
- Stack pointer (SP)
- Program counter (PC)
  Address of currently executed instruction

[source: [ColdFire Family Programmers Reference]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example: Freescale ColdFire (1)



Freescale ColdFire

- 32 bit architecture
- 16 general purpose 32 bit registers (D0-D7, A0-A7)
- 8 bit condition code register/16 bit status register

Address registers

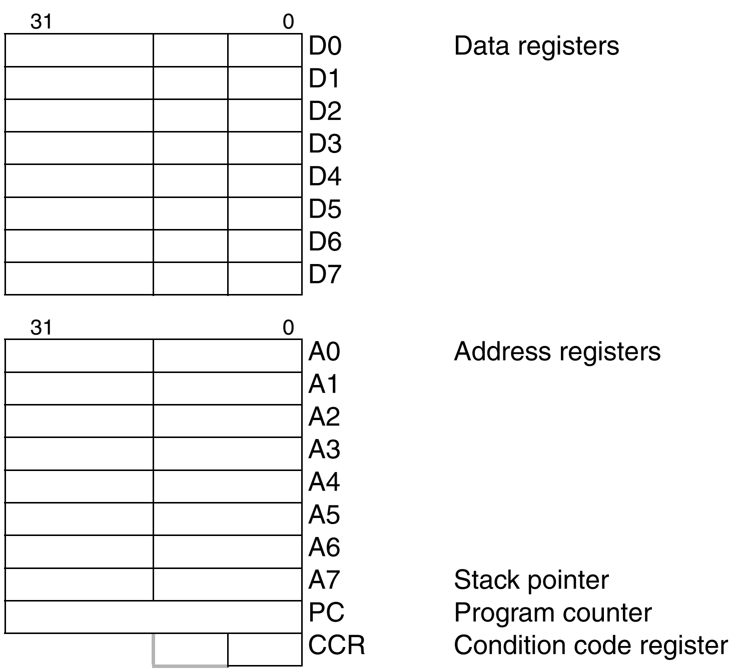
- Addresses of variables
- Base address registers
- Index address registers
- Stack pointer (SP)
- Program counter (PC)
  Address of currently executed instruction

[source: [ColdFire Family Programmers Reference]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example: Freescale ColdFire (1)



Freescale ColdFire

- 32 bit architecture
- 16 general purpose 32 bit registers (D0-D7, A0-A7)
- 8 bit condition code register/16 bit status register

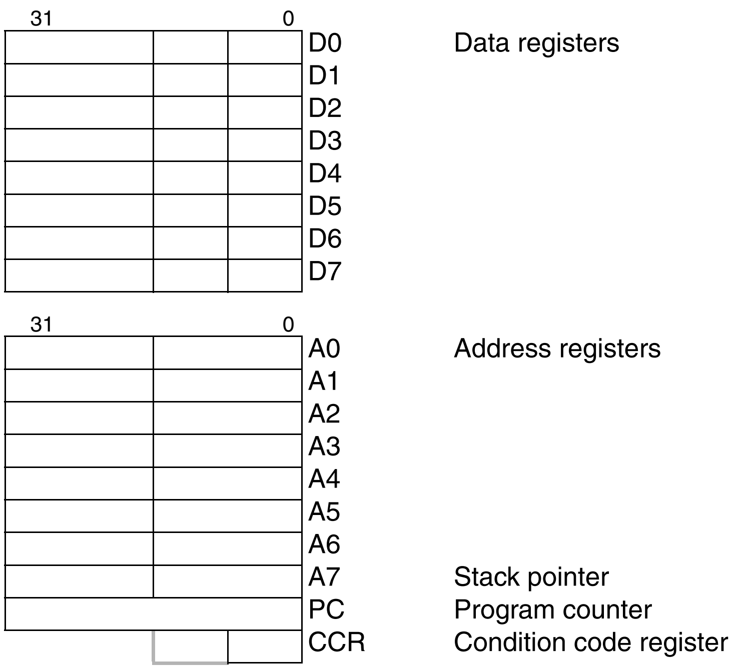
Address registers

- Addresses of variables
- Base address registers
- Index address registers
- Stack pointer (SP)
- Program counter (PC)
  Address of currently executed instruction

[source: [ColdFire Family Programmers Reference]]

CAMPUS Rosenheim
Computer Science

Example: Freescale ColdFire (1)



Freescale ColdFire

- 32 bit architecture
- 16 general purpose 32 bit registers (D0-D7, A0-A7)
- 8 bit condition code register/16 bit status register

Address registers

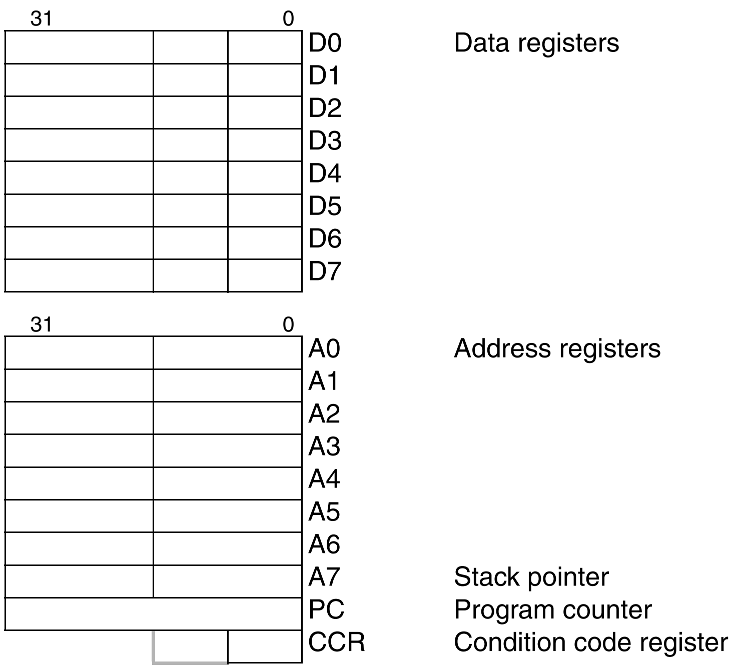
- Addresses of variables
- Base address registers
- Index address registers
- Stack pointer (SP)
- Program counter (PC)
  Address of currently executed instruction

[source: [ColdFire Family Programmers Reference]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example: Freescale ColdFire (1)



Freescale ColdFire

- 32 bit architecture
- 16 general purpose 32 bit registers (D0-D7, A0-A7)
- 8 bit condition code register/16 bit status register

Address registers

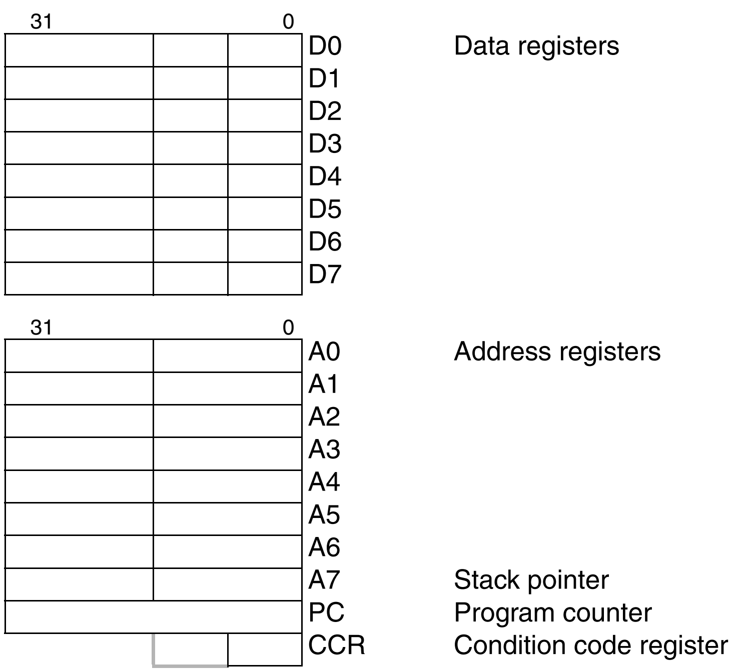
- Addresses of variables
- Base address registers
- Index address registers
- Stack pointer (SP)
- Program counter (PC)
  Address of currently executed instruction

[source: [ColdFire Family Programmers Reference]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example: Freescale ColdFire (1)



Freescale ColdFire

- 32 bit architecture
- 16 general purpose 32 bit registers (D0-D7, A0-A7)
- 8 bit condition code register/16 bit status register

Address registers

- Addresses of variables
- Base address registers
- Index address registers
- Stack pointer (SP)
- Program counter (PC)
  Address of currently executed instruction

[source: [ColdFire Family Programmers Reference]]

Example: Freescale ColdFire (2)

Condition Code Register (CCR)

7	6–5	4	3	2	1	0
P[1]	—	X	N	Z	V	C

[1]The P bit is implemented only on the V3 core.

Figure 1-2. Condition Code Register (CCR)

Table 1-1 describes CCR bits.

Table 1-1. CCR Bit Descriptions

Bits	Field	Description
7	P	Branch prediction (Version 3 only). Alters the static prediction algorithm used by the branch acceleration logic in the instruction fetch pipeline on forward conditional branches. Refer to a V3 core or device user's manual for further information on this bit.
	—	Reserved; should be cleared (all other versions).
6–5	—	Reserved, should be cleared.
4	X	Extend. Set to the value of the C-bit for arithmetic operations; otherwise not affected or set to a specified result.
3	N	Negative. Set if the most significant bit of the result is set; otherwise cleared.
2	Z	Zero. Set if the result equals zero; otherwise cleared.
1	V	Overflow. Set if an arithmetic overflow occurs implying that the result cannot be represented in the operand size; otherwise cleared.
0	C	Carry. Set if a carry out of the most significant bit of the operand occurs for an addition, or if a borrow occurs in a subtraction; otherwise cleared.

[source: [ColdFire Family Programmers Reference]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Example: Freescale ColdFire (3)
Status Register (SR)

15	14	13	12	11	10 … 8	7	6 … 5	4	3	2	1	0
System byte						Condition code register (CCR)						
T	—	S	M	—	I	P[1]	—	X	N	Z	V	C

[1]The P bit is implemented only on the V3 core.

Figure 1-15. Status Register (SR)

Table 1-7 describes SR fields.

Table 1-7. Status Field Descriptions

Bits	Name	Description
15	T	Trace enable. When T is set, the processor performs a trace exception after every instruction.
14	—	Reserved, should be cleared.
13	S	Supervisor/user state. Indicates whether the processor is in supervisor or user mode
12	M	Master/interrupt state. Cleared by an interrupt exception. It can be set by software during execution of the RTE or move to SR instructions so the OS can emulate an interrupt stack pointer.
11	—	Reserved, should be cleared.
10–8	I	Interrupt priority mask. Defines the current interrupt priority. Interrupt requests are inhibited for all priority levels less than or equal to the current priority, except the edge-sensitive level-7 request, which cannot be masked.
7–0	CCR	Condition code register (see Figure 1-2 and Table 1-1)

[source: [ColdFire Family Programmers Reference]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Intel 32 bit (x86) (1)

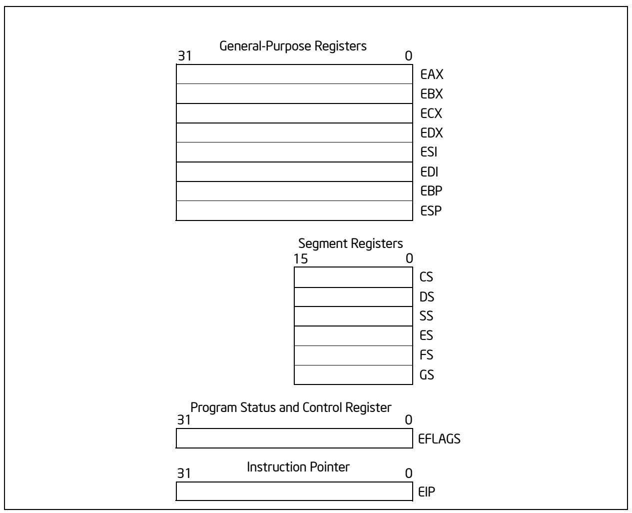


Figure 3-4. General System and Application Programming Registers

Register names

- EAX – Accumulator for operands and results data
- EBX – Pointer to data in the DS segment
- ECX – Counter for string and loop operations
- EDX – I/O pointer
- ESI – Pointer to data in the segment (DS register)
- EDI – Pointer to data in the segment (ES register)
- EBP – Pointer to data on the stack (in the SS segment)
- ESP – Stack pointer (in the SS segment)

Extended

[source: [Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, p. 3-11]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Intel 32 bit (x86) (2)

L=low
H=HIGH

AX

General-Purpose Registers

31 … 16	15 … 8	7 … 0	**16-bit**	**32-bit**
	AH	AL	AX	EAX
	BH	BL	BX	EBX
	CH	CL	CX	ECX
	DH	DL	DX	EDX
	BP			EBP
	SI			ESI
	DI			EDI
	SP			ESP

Figure 3-5. Alternate General-Purpose Register Names

[source: [Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, p. 3-12]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Intel 32 bit (x86) (3)

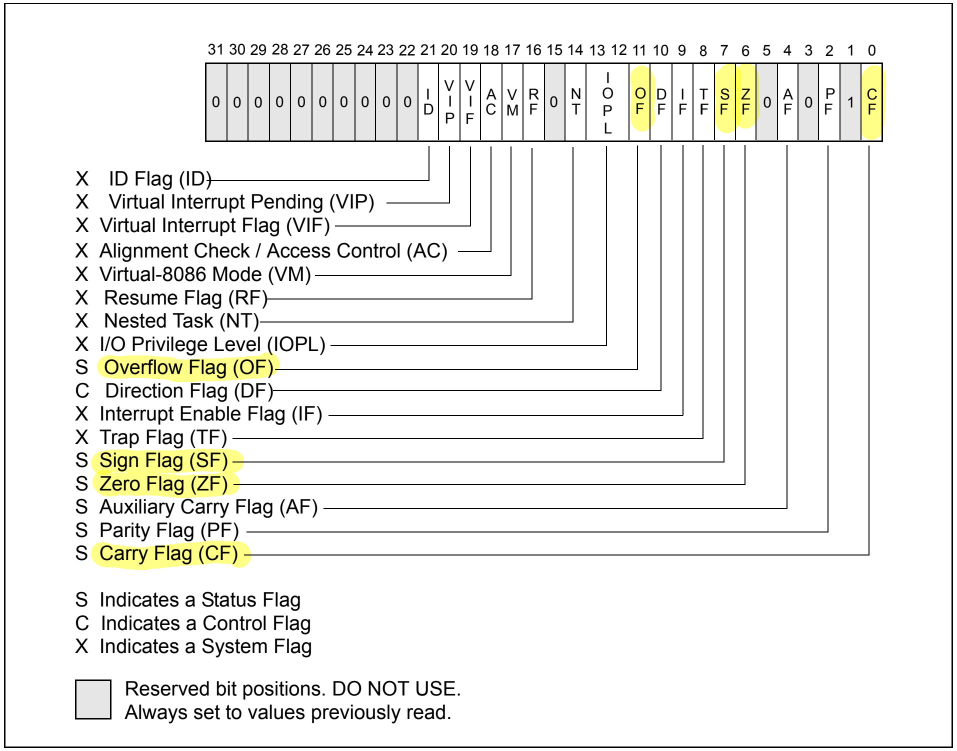


Figure 3-8. EFLAGS Register

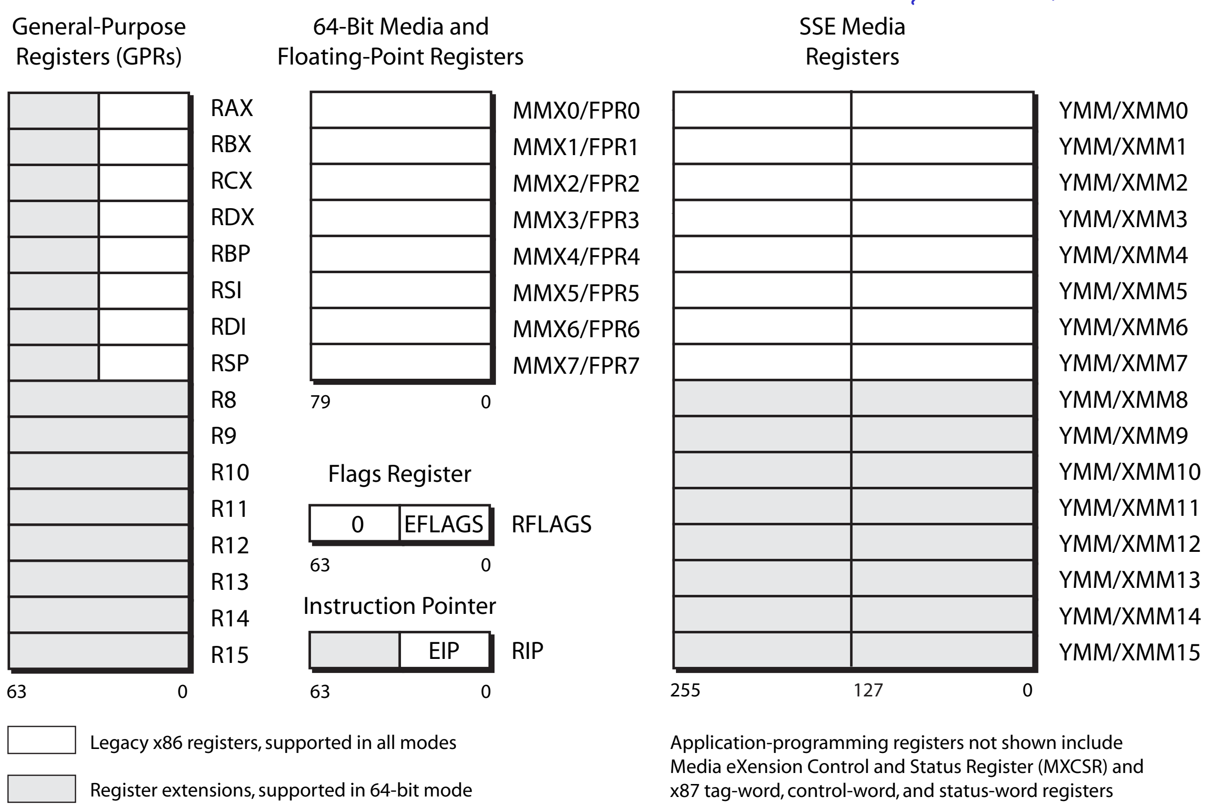
[source: [Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1, p. 3-16]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Intel 64 bit (x86-64) (1)

- Streaming SIMD Extension

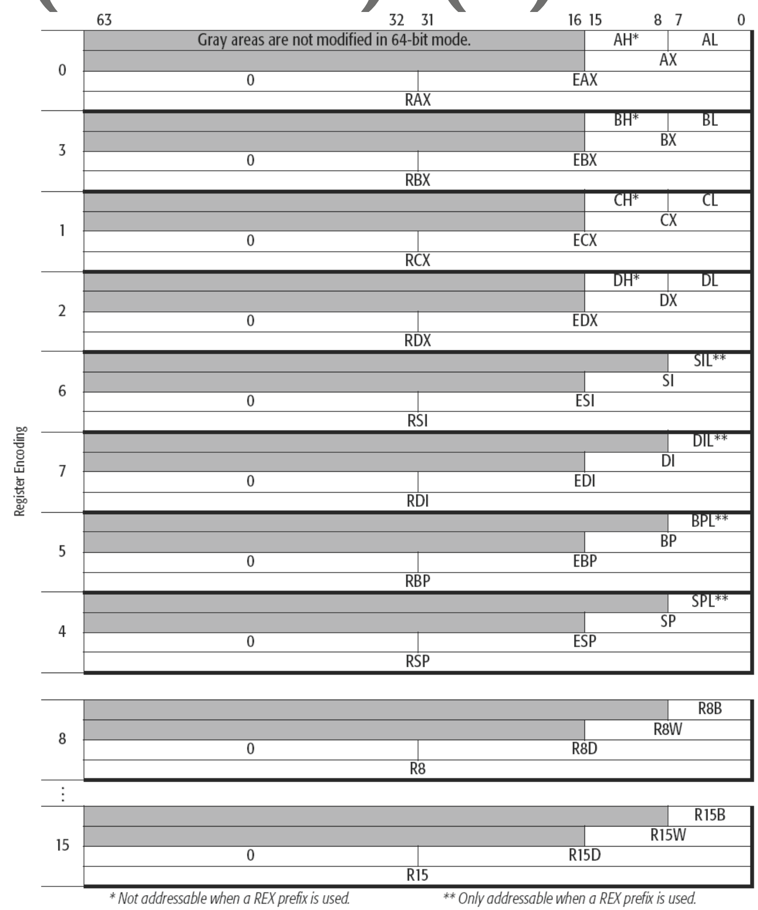


[source: [AMD64 Architecture Programmer's Manual, Volume 1: Application Programming, p. 2]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Intel 64 bit (x86-64) (2)



[source: [AMD64 Architecture Programmer's Manual, Volume 1: Application Programming, p. 28]]

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Instructions

The set of all machine instructions of a processor is called **instruction set**.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Instruction overview 1/3

Overview of the basic **intel x86** **instruction set** groups.

1. Move instructions

`MOV`	move/transfer operands
`IN`	peripheral input
`OUT`	peripheral output
`PUSH`	write register to stack
`POP`	read register from stack
`PUSHF`	write flag register to stack
`POPF`	read flag register from stack

...

2. Arithmetic instructions

`ADD`	add
`SUB`	substract
`INC`	increase by 1
`DEC`	decrease by 1
`CMP`	compare
`NEG`	negate
`MUL`	multiply
`DIV`	divide

...

Intel x86 instruction set groups according to "Informatik für Ingenieure und Naturwissenschaftler 2" page 67.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Instruction overview 2/3

Overview of the basic **intel x86 instruction set** groups.

3. Logic instructions

AND	logical AND
OR	logical OR
XOR	exclusive OR
NOT	not (ones' complement)
SHR	logical right shift
SHL	logical left shift
ROR	arithmetic right shift
ROL	arithmetic left shift

...

4. Jump instructions

JMP	unconditional jump
JG	conditional jump: if greater
JNZ	conditional jump: if not equal 0
LOOP	conditional jump: if CX $\neq$ 0
CALL	call function
RET	return from function
INT	software interrupt
IRET	return from ISR

...

Intel x86 instruction set groups according to "Informatik für Ingenieure und Naturwissenschaftler 2" page 67.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Instruction overview 3/3

Overview of the basic **intel x86 instruction set** groups.

5. Processor control instructions

`HLT`	halt (stop) processor
`STC`	set carry flag
`CLC`	clear carry flag
`SYSCALL`	system call
`SYSRET`	system return

...

6. String instructions

`MOVS`	move string
`LODS`	load string
`STOS`	store string
`CMPS`	compare string
`SCAS`	scan string
`REP`	repeat while %ecx not zero
`REPE`	repeat while not equal

...

Intel x86 instruction set groups according to "Informatik für Ingenieure und Naturwissenschaftler 2" page 67.
Some links for string instructions: [1] [2]

CAMPUS Rosenheim
Computer Science

(A0) = 0x370 0x3F0

index · word width = Xi

		Bits
.B	Byte	8
.W	Word	16
.L	Long	32
.Q	Quad	64

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing modes

	CISC Freescale		CISC Intel	RISC Cortex-M0
Mode	**Syntax**	**Operand address**		
Register direct	`Dn or. An`	`Dn or. An`	`AL, AH, EAX, R8`	`Rn`
Immediate	`#xxx`	data = (next word(s))	`XXX`	`#xxx`
Memory direct short	`xxx.W`	(next 16-bit-word)	`[word XXX]`	
Memory direct long	`xxx.L`	(next 32-bit-word)	`[dword XXX]`	
HW Pointer Register indirect	`(An)`	Content of An	`[EBX]`	`[R1]`
Register indirect post-increment	`(An)+`	`(An); An = An + N`		`[R1], #4`
Register indirect pre-decrement	`-(An)`	`An= An - N; (An)`		
Register indirect displacement	`(d, An)`	`d + (An)`	`[EBX]d`	`[R1, #4]`
Register indirect indexed	`(d, An, Xi)`	`d + (An) + (Xi)`	`[EBX+SI]d`	`[R1, R2]`

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing modes

	Freescale		Intel	Cortex-M0
Mode	**Syntax**	**Operand address**		
Memory indirect post-indexed	`([d, An] ...) ...`	`(Memory[d+(An)]) ...`		
Memory indirect pre-indexed	`([d, An, ...]) ...`	`(Memory[d+(An) ...])`		
Program counter relative Stack pointer	`(d, PC)` (d, SP)	`d + (PC)`		`[PC, #offset]`
Program counter indexed	`(d, PC, Xi)`	`d + (PC) + (Xi)`		
PC memory indirect post-indexed	...	...		
PC memory indirect pre-indexed	...	...		

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Instruction format (principle)

Format of the addressing information in the machine commands (principle):

- X bits for opcode
- 3 bits for specifying a register
- 3 bits for specifying an addressing mode

Example: `MOV R1, R2 ;move R2 to R1`

Encoding

Opcode	Instruction	Code for mode	addressing mode
...0000	ADD	000	Register direct
...0001	MOV	001	Register indirect
...0010	SUB	010	Register indirect post-increment
...0011	MUL	011	Register indirect pre-decrement
...0100	CMP	100	Register indirect displacement

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Instruction format (principle)

Format of the addressing information in the machine commands (principle):

- X bits for opcode
- 3 bits for specifying a register
- 3 bits for specifying an addressing mode

Example: `MOV R1, R2 ;move R2 to R1`

Encoding

Opcode	Instruction	Code for mode	addressing mode
...0000	ADD	000	Register direct
...0001	MOV	001	Register indirect
...0010	SUB	010	Register indirect post-increment
...0011	MUL	011	Register indirect pre-decrement
...0100	CMP	100	Register indirect displacement

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Instruction format (principle)

Format of the addressing information in the machine commands (principle):

- X bits for opcode
- 3 bits for specifying a register
- 3 bits for specifying an addressing mode

Example: `MOV R1, R2 ;move R2 to R1`

Encoding

Opcode	Instruction	Code for mode	addressing mode
...0000	ADD	000	Register direct
...0001	MOV	001	Register indirect
...0010	SUB	010	Register indirect post-increment
...0011	MUL	011	Register indirect pre-decrement
...0100	CMP	100	Register indirect displacement

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Instruction format (principle)

Format of the addressing information in the machine commands (principle):

- X bits for opcode
- 3 bits for specifying a register
- 3 bits for specifying an addressing mode

Example: `MOV R1, R2 ;move R2 to R1`

Encoding

Opcode	Instruction	Code for mode	addressing mode
...0000	ADD	000	Register direct
...0001	MOV	001	Register indirect
...0010	SUB	010	Register indirect post-increment
...0011	MUL	011	Register indirect pre-decrement
...0100	CMP	100	Register indirect displacement

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Instruction format (principle)

Format of the addressing information in the machine commands (principle):

- X bits for opcode
- 3 bits for specifying a register
- 3 bits for specifying an addressing mode

Example: `MOV R1, R2 ;move R2 to R1`

… 0 0 0 1	0 0 1	0 0 0	0 1 0	0 0 0
Opcode	Register	Mode	Register	Mode
	Destination		Source	

Encoding

Opcode	Instruction
…0000	ADD
…0001	MOV
…0010	SUB
…0011	MUL
…0100	CMP

Code for mode	addressing mode
000	Register direct
001	Register indirect
010	Register indirect post-increment
011	Register indirect pre-decrement
100	Register indirect displacement

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Instruction format (principle)

Format of the addressing information in the machine commands (principle):

- X bits for opcode
- 3 bits for specifying a register
- 3 bits for specifying an addressing mode

Example: `MOV R1, R2 ;move R2 to R1`

Opcode	Register	Mode	Register	Mode
… 0 0 0 1	0 0 1	0 0 0	0 1 0	0 0 0
	Destination		Source	

Encoding

Opcode	**Instruction**	**Code for mode**	**addressing mode**
...0000	ADD	000	Register direct
...0001	MOV	001	Register indirect
...0010	SUB	010	Register indirect post-increment
...0011	MUL	011	Register indirect pre-decrement
...0100	CMP	100	Register indirect displacement

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing modes

Orthogonal architecture:

If all operations can be **combined with all types of addressing** (as far as it makes sense), such an architectural design is called **orthogonal** (usually only in CISC architectures).

Real world:

In practice, however, most architectures have more or less unpleasant exceptions and limitations.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing modes

Orthogonal architecture:

If all operations can be **combined with all types of addressing** (as far as it makes sense), such an architectural design is called **orthogonal** (usually only in CISC architectures).

Real world:

In practice, however, most architectures have more or less unpleasant exceptions and limitations.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (1)

Register indirect post-increment

Example: Copy a memory area

C code

```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
    #define STR_LEN (5)
    char s1[STR_LEN] = "...";  //source
    char s2[STR_LEN];          //destination

    //copy a memory area (strings)
    strncpy(s2, s1, STR_LEN);

    printf("%s\n", s2);

    return EXIT_SUCCESS;
}
```

Assembler code*

```
 ;strncopy (simplified) asm example
 MOVE 0x..., A1 ;copy address of s1 to A1
 MOVE 0x..., A2 ;copy address of s2 to A2

;some loop (simplified):
;for(i = 0; i < STR_LEN; ++i)
    MOVE.B (A1)+, (A2)+  ;copy char by char
```

*This is pseudo assembler code (somehow based on Freescale ColdFire)

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (1)

Register indirect post-increment

Example: Copy a memory area

C code

```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
    #define STR_LEN (5)
    char s1[STR_LEN] = "..."; //source
    char s2[STR_LEN];         //destination

    //copy a memory area (strings)
    strncpy(s2, s1, STR_LEN);

    printf("%s\n", s2);

    return EXIT_SUCCESS;
}
```

Assembler code*

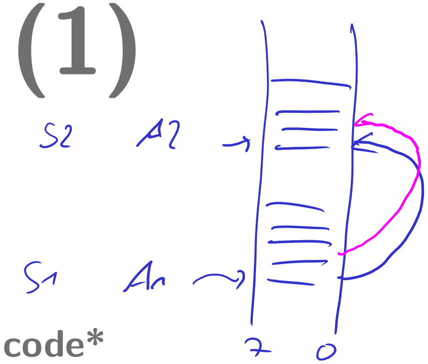
```
 ;strncopy (simplified) asm example
 MOVE 0x..., A1 ;copy address of s1 to A1
 MOVE 0x..., A2 ;copy address of s2 to A2

;some loop (simplified):
;for(i = 0; i < STR_LEN; ++i)
    MOVE.B (A1)+, (A2)+  ;copy char by char
```

**This is pseudo assembler code (somehow based on Freescale ColdFire)*

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (1)



Register indirect post-increment

Example: Copy a memory area

C code

```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main(void)
{
    #define STR_LEN (5)
    char s1[STR_LEN] = "..."; //source
    char s2[STR_LEN];         //destination

    //copy a memory area (strings)
    strncpy(s2, s1, STR_LEN);

    printf("%s\n", s2);

    return EXIT_SUCCESS;
}
```

Assembler code*

```
 ;strncopy (simplified) asm example
 MOVE 0x..., A1 ;copy address of s1 to A1
 MOVE 0x..., A2 ;copy address of s2 to A2

;some loop (simplified):
;for(i = 0; i < STR_LEN; ++i)
    MOVE.B (A1)+, (A2)+  ;copy char by char
```

**This is pseudo assembler code (somehow based on Freescale ColdFire)*

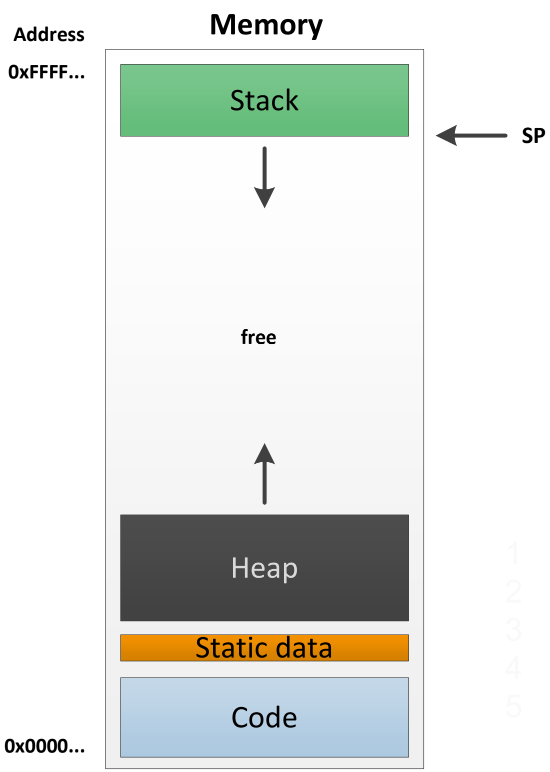
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (2)

Register indirect pre-decrement and post-increment

Example: Stack operations



Stack properties

- Is by convention in a normal memory area
- Growth towards lower memory addresses
- The SP (stack pointer) points to the last used address
- Last in first out (LIFO) principle
- Very simple and typically faster than heap-based memory allocation

Stack usage

- Return addresses and register contents
- Function and local parameters

Assembler code*

```
PUSH_X:
    MOVE.W X, -(SP)   ; move word X to stack

POP_Y:
    MOVE.W (SP)+, Y   ; move word from stack to Y
```

*This is pseudo assembler code (somehow based on Freescale ColdFire)

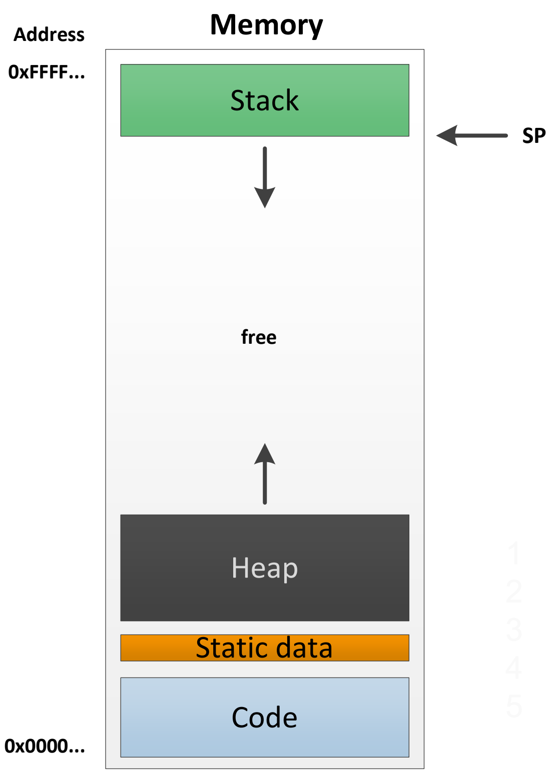
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (2)

Register indirect pre-decrement and post-increment

Example: Stack operations



Stack properties

- Is by convention in a normal memory area
- Growth towards lower memory addresses
- The SP (stack pointer) points to the last used address
- Last in first out (LIFO) principle
- Very simple and typically faster than heap-based memory allocation

Stack usage

- Return addresses and register contents
- Function and local parameters

Assembler code*

```
PUSH_X:
    MOVE.W X, -(SP)   ; move word X to stack

POP_Y:
    MOVE.W (SP)+, Y   ; move word from stack to Y
```

*This is pseudo assembler code (somehow based on Freescale ColdFire)

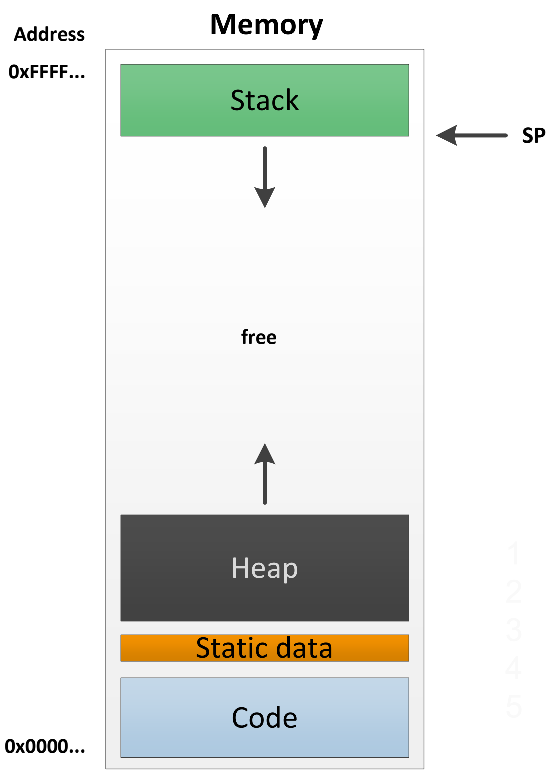
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (2)

Register indirect pre-decrement and post-increment

Example: Stack operations



Stack properties

- Is by convention in a normal memory area
- Growth towards lower memory addresses
- The SP (stack pointer) points to the last used address
- Last in first out (LIFO) principle
- Very simple and typically faster than heap-based memory allocation

Stack usage

- Return addresses and register contents
- Function and local parameters

Assembler code*

```
PUSH_X:
    MOVE.W X, -(SP)   ; move word X to stack

POP_Y:
    MOVE.W (SP)+, Y   ; move word from stack to Y
```

*This is pseudo assembler code (somehow based on Freescale ColdFire)

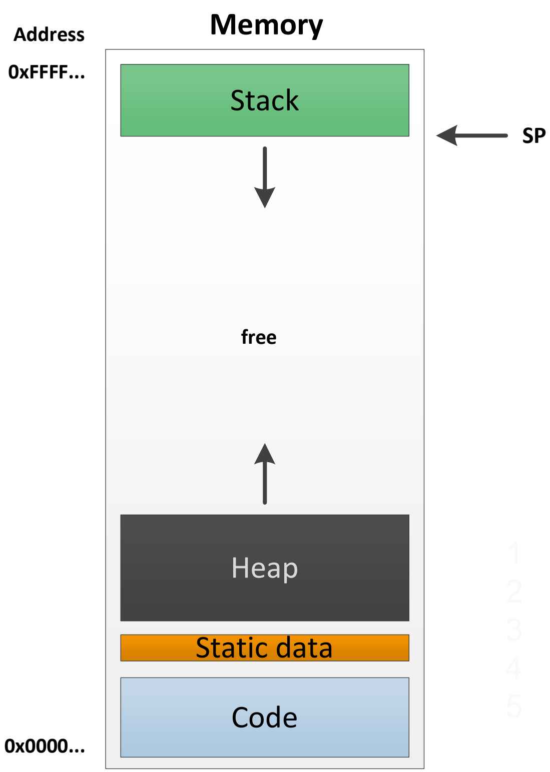
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (2)

Register indirect pre-decrement and post-increment

Example: Stack operations



Stack properties

- Is by convention in a normal memory area
- Growth towards lower memory addresses
- The SP (stack pointer) points to the last used address
- Last in first out (LIFO) principle
- Very simple and typically faster than heap-based memory allocation

Stack usage

- Return addresses and register contents
- Function and local parameters

Assembler code*

```
PUSH_X:
    MOVE.W X, -(SP)   ; move word X to stack

POP_Y:
    MOVE.W (SP)+, Y   ; move word from stack to Y
```

*This is pseudo assembler code (somehow based on Freescale ColdFire)

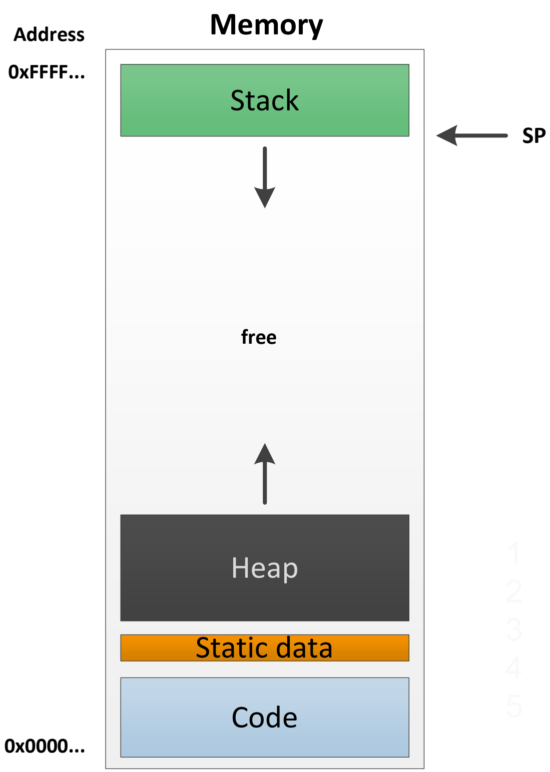
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (2)

Register indirect pre-decrement and post-increment

Example: Stack operations



Stack properties

- Is by convention in a normal memory area
- Growth towards lower memory addresses
- The SP (stack pointer) points to the last used address
- Last in first out (LIFO) principle
- Very simple and typically faster than heap-based memory allocation

Stack usage

- Return addresses and register contents
- Function and local parameters

Assembler code*

```
PUSH_X:
    MOVE.W X, -(SP)   ; move word X to stack

POP_Y:
    MOVE.W (SP)+, Y   ; move word from stack to Y
```

*This is pseudo assembler code (somehow based on Freescale ColdFire)

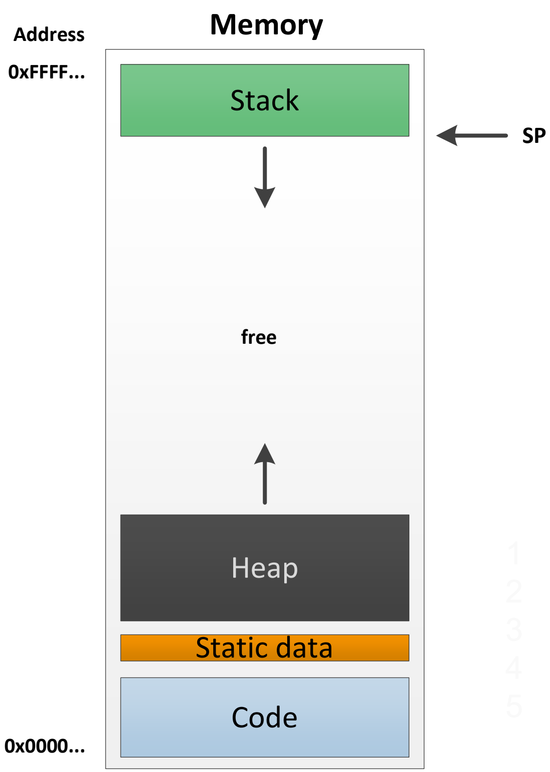
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (2)

Register indirect pre-decrement and post-increment

Example: Stack operations



Stack properties

- Is by convention in a normal memory area
- Growth towards lower memory addresses
- The SP (stack pointer) points to the last used address
- Last in first out (LIFO) principle
- Very simple and typically faster than heap-based memory allocation

Stack usage

- Return addresses and register contents
- Function and local parameters

Assembler code*

```
PUSH_X:
    MOVE.W X, -(SP)   ; move word X to stack

POP_Y:
    MOVE.W (SP)+, Y   ; move word from stack to Y
```

*This is pseudo assembler code (somehow based on Freescale ColdFire)

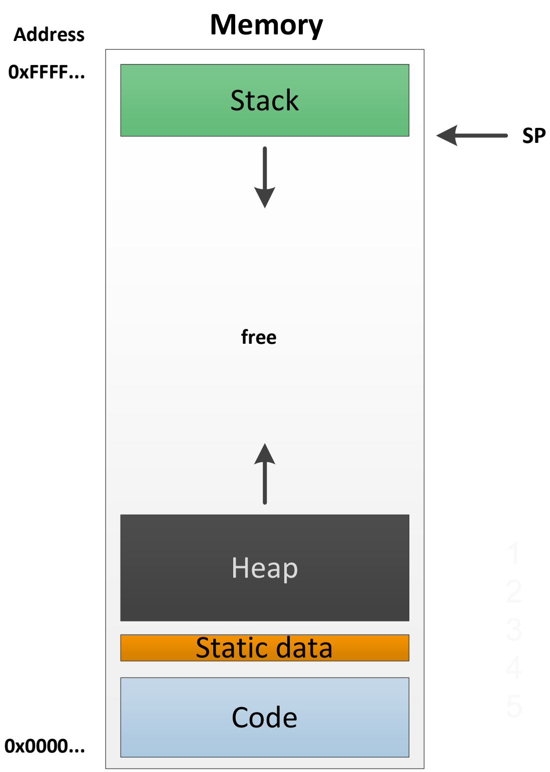
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (2)

Register indirect pre-decrement and post-increment

Example: Stack operations



Stack properties

- Is by convention in a normal memory area
- Growth towards lower memory addresses
- The SP (stack pointer) points to the last used address
- Last in first out (LIFO) principle
- Very simple and typically faster than heap-based memory allocation

Stack usage

- Return addresses and register contents
- Function and local parameters

Assembler code*

```
PUSH_X:
    MOVE.W X, -(SP)   ; move word X to stack

POP_Y:
    MOVE.W (SP)+, Y   ; move word from stack to Y
```

*This is pseudo assembler code (somehow based on Freescale ColdFire)

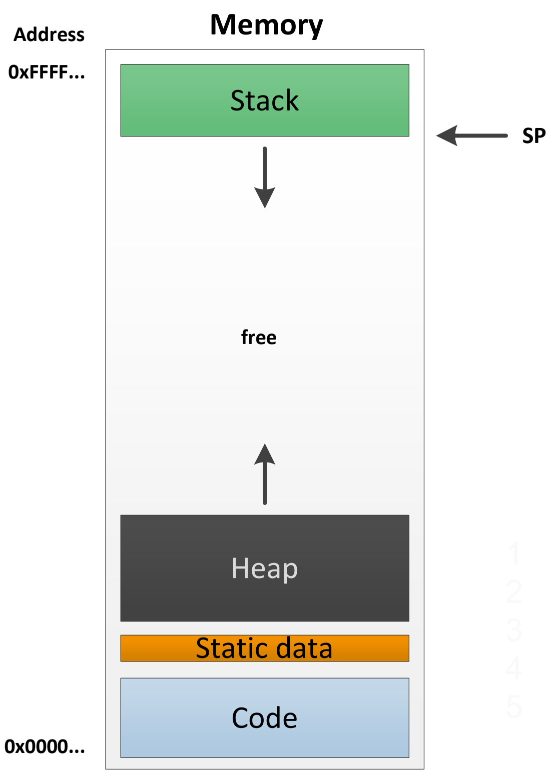
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (2)

Register indirect pre-decrement and post-increment

Example: Stack operations



Stack properties

- Is by convention in a normal memory area
- Growth towards lower memory addresses
- The SP (stack pointer) points to the last used address
- Last in first out (LIFO) principle
- Very simple and typically faster than heap-based memory allocation

Stack usage

- Return addresses and register contents
- Function and local parameters

Assembler code*

```
PUSH_X:
    MOVE.W X, -(SP)   ; move word X to stack

POP_Y:
    MOVE.W (SP)+, Y   ; move word from stack to Y
```

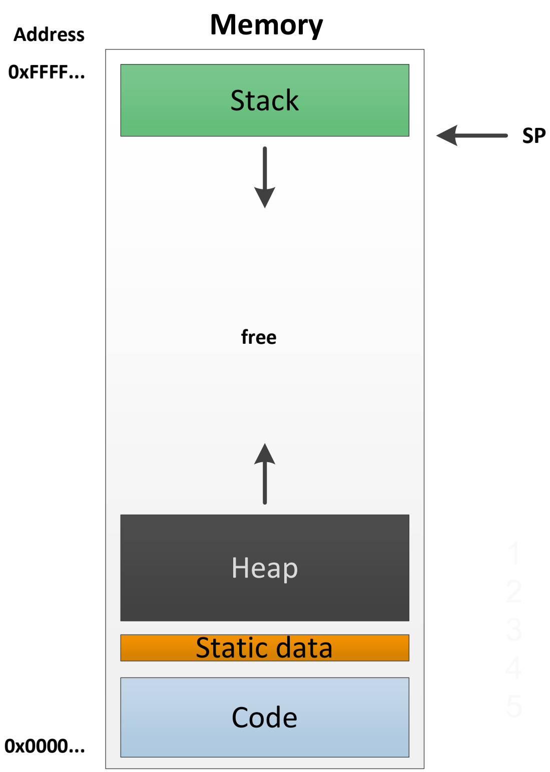
*This is pseudo assembler code (somehow based on Freescale ColdFire)

CAMPUS Rosenheim
Computer Science

Addressing mode example (2)

Register indirect pre-decrement and post-increment

Example: Stack operations



Stack properties

- Is by convention in a normal memory area
- Growth towards lower memory addresses
- The SP (stack pointer) points to the last used address
- Last in first out (LIFO) principle
- Very simple and typically faster than heap-based memory allocation

Stack usage

- Return addresses and register contents
- Function and local parameters

Assembler code*

```
PUSH_X:
    MOVE.W X, -(SP)   ; move word X to stack

POP_Y:
    MOVE.W (SP)+, Y   ; move word from stack to Y
```

*This is pseudo assembler code (somehow based on Freescale ColdFire)

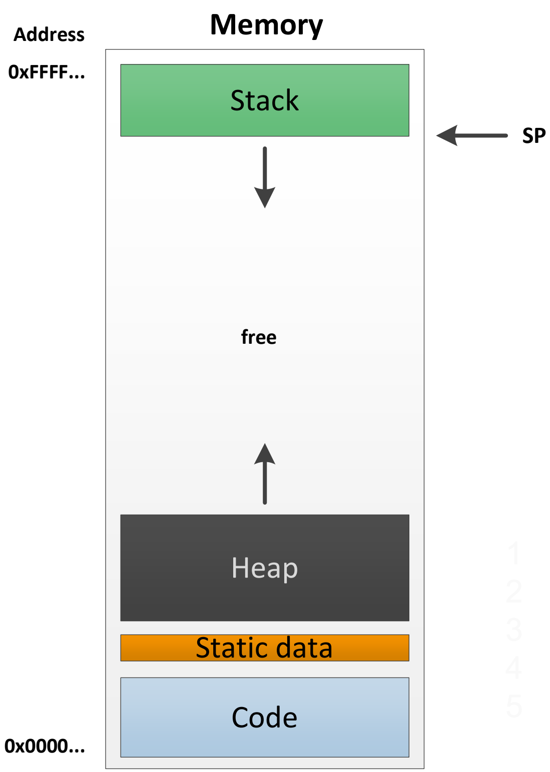
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (2)

Register indirect pre-decrement and post-increment

Example: Stack operations



Stack properties

- Is by convention in a normal memory area
- Growth towards lower memory addresses
- The SP (stack pointer) points to the last used address
- Last in first out (LIFO) principle
- Very simple and typically faster than heap-based memory allocation

Stack usage

- Return addresses and register contents
- Function and local parameters

Assembler code*

```
PUSH_X:
    MOVE.W X, -(SP)   ; move word X to stack

POP_Y:
    MOVE.W (SP)+, Y   ; move word from stack to Y
```

*This is pseudo assembler code (somehow based on Freescale ColdFire)

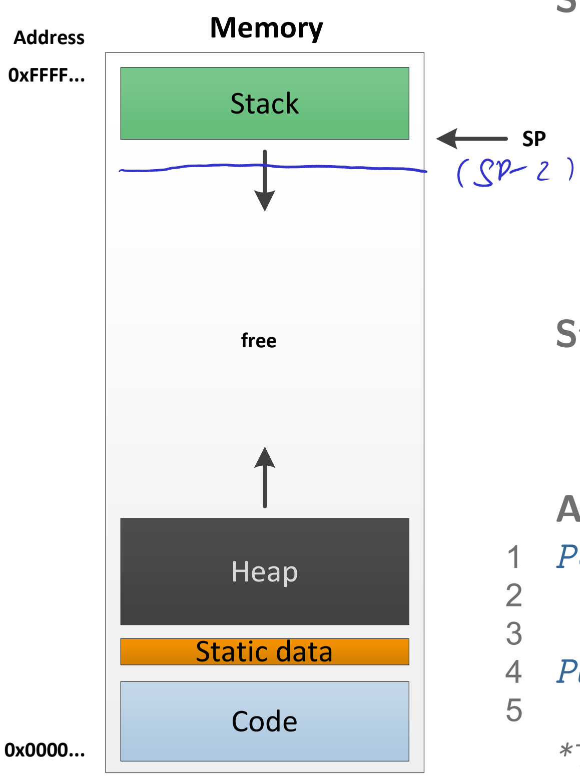
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (2)

Register indirect pre-decrement and post-increment

Example: Stack operations



Stack properties

- Is by convention in a normal memory area
- Growth towards lower memory addresses
- The SP (stack pointer) points to the last used address
- Last in first out (LIFO) principle
- Very simple and typically faster than heap-based memory allocation

Stack usage

- Return addresses and register contents
- Function and local parameters

Assembler code*

```
PUSH_X:
    MOVE.W X, -(SP)   ; move word X to stack

POP_Y:
    MOVE.W (SP)+, Y   ; move word from stack to Y
```

**This is pseudo assembler code (somehow based on Freescale ColdFire)*

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (3)

Register indirect indexed

Example: Access an array element

C code

```c
#include <stdlib.h>

int main(void)
{
    const int NUM_VALUES = 100;
    int values[NUM_VALUES];

    //do something useful
    int i = 5;
    values[i] = 123;

    return EXIT_SUCCESS;
}
```

Assembler code*

```
 ;array access (simplified) asm example
 MOVE 0x..., A1 ;copy address of values to A1
 MOVE i,     D1 ;copy content of i into D1

;vals[i] = 123;
    MOVE.L #123, (0, A1, D1*4)
```

**This is pseudo assembler code (somehow based on Freescale ColdFire)*

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (3)

Register indirect indexed

Example: Access an array element

C code

```c
#include <stdlib.h>

int main(void)
{
    const int NUM_VALUES = 100;
    int values[NUM_VALUES];

    //do something useful
    int i = 5;
    values[i] = 123;

    return EXIT_SUCCESS;
}
```

Assembler code*

```
 ;array access (simplified) asm example
 MOVE 0x..., A1 ;copy address of values to A1
 MOVE i,     D1 ;copy content of i into D1

;vals[i] = 123;
    MOVE.L #123, (0, A1, D1*4)
```

**This is pseudo assembler code (somehow based on Freescale ColdFire)*

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (3)

Register indirect indexed

Example: Access an array element

C code

```c
#include <stdlib.h>

int main(void)
{
    const int NUM_VALUES = 100;
    int values[NUM_VALUES];

    //do something useful
    int i = 5;
    values[i] = 123;

    return EXIT_SUCCESS;
}
```

Assembler code*

```
 ;array access (simplified) asm example
 MOVE 0x..., A1 ;copy address of values to A1
 MOVE i,     D1 ;copy content of i into D1

;vals[i] = 123;
    MOVE.L #123, (0, A1, D1*4)
```

**This is pseudo assembler code (somehow based on Freescale ColdFire)*

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (4)

Program counter relative

Example: Position independent code (PIC)

- Shared libraries
- Processes on a CPU without MMU (virtual memory)

C code

```c
#include <stdlib.h>

int main(void)
{
    int value;
    value = 0xABCDEF01;

    return EXIT_SUCCESS;
}
```

Assembler code*

```
;value = 0xABCDEF01;
    MOVE.L (#4, PC), (#0, SP)
    0xABCDEF01
    ;... next instruction
```

*This is pseudo assembler code (somehow based on Freescale ColdFire, but not tested)

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (4)

Program counter relative

Example: Position independent code (PIC)

- Shared libraries
- Processes on a CPU without MMU (virtual memory)

C code

```c
#include <stdlib.h>

int main(void)
{
    int value;
    value = 0xABCDEF01;

    return EXIT_SUCCESS;
}
```

Assembler code*

```
;value = 0xABCDEF01;
    MOVE.L (#4, PC), (#0, SP)
    0xABCDEF01
    ;... next instruction
```

**This is pseudo assembler code (somehow based on Freescale ColdFire, but not tested)*

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Addressing mode example (4)

Program counter relative

Example: Position independent code (PIC)

- Shared libraries
- Processes on a CPU without MMU (virtual memory)

C code

```c
#include <stdlib.h>

int main(void)
{
    int value;
    value = 0xABCDEF01;

    return EXIT_SUCCESS;
}
```

Assembler code*

```
;value = 0xABCDEF01;
    MOVE.L (#4, PC), (#0, SP)
    0xABCDEF01
    ;... next instruction
```

**This is pseudo assembler code (somehow based on Freescale ColdFire, but not tested)*

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Comparison

Compare different instruction set architectures!

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

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Summary and outlook

Summary

- Registers
- Processor examples
- Addressing modes

Outlook

- Pipelining
- Instruction scheduling
- Superscalar architecture
- VLIW
- Out-of-order memory access

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Summary and outlook

Summary

- Registers
- Processor examples
- Addressing modes

Outlook

- Pipelining
- Instruction scheduling
- Superscalar architecture
- VLIW
- Out-of-order memory access