System Construction

Autumn Semester 2019

ETH Zürich

Felix Friedrich, Paul Reed

Goals

- Competence in building custom system software **from scratch**
- Understanding of „how it really works“ behind the scenes **across all levels**
- Knowledge of the approach of fully managed **simple** systems

A lot of this course **is about detail.**
A lot of this course is about **bare metal programming**.

Course Concept

- Discussing elaborated case studies
  - In theory (lectures)
  - and **practice** (hands-on lab)
- Learning by example vs. presenting topics

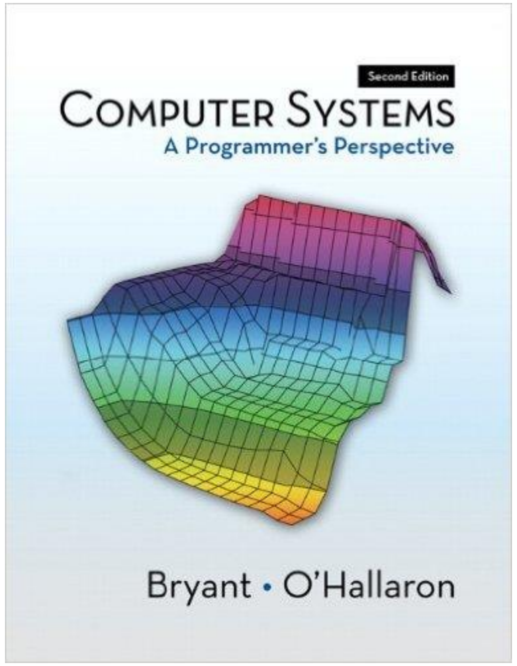
Prerequisites

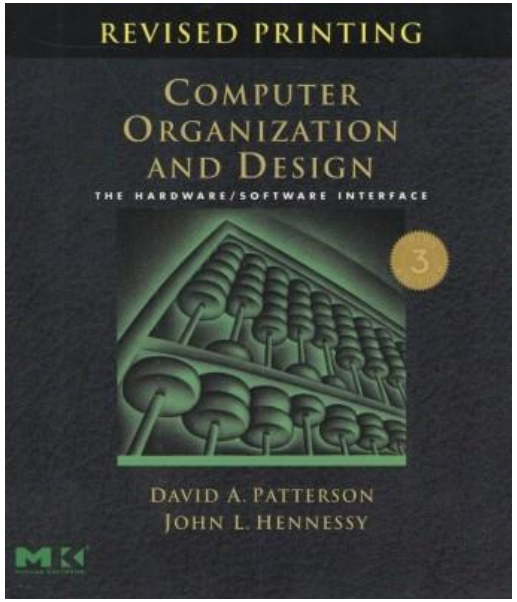
Knowledge corresponding to lectures
Systems Programming [and Computer Architecture]

- *Do you know what a stack-frame is?*
- *Do you know how an interrupt works?*
- *Do you know the concept of virtual memory?*

Good references for recapitulation:

- Randal E. Bryant, David Richard O'Hallaron, *Computer Systems – A Programmer's Perspective,*
- David A. Patterson, John L. Hennessy *Computer Organization and Design – The Hardware/Software Interface ,*





Links

- SVN repository

https://svn.inf.ethz.ch/svn/lecturers/vorlesungen/trunk/syscon/2019/shared

- Links on the course homepage

http://lec.inf.ethz.ch/syscon

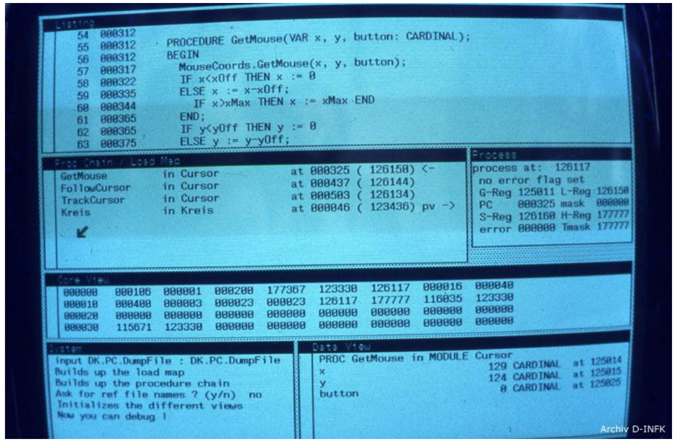
Some ETH History

1980: Niklaus Wirth develops Lilith, one of the first computers with graphical user interface: bitmap display and mouse

Lilith was constructed from 4-bit AMD-Am2900 Slices

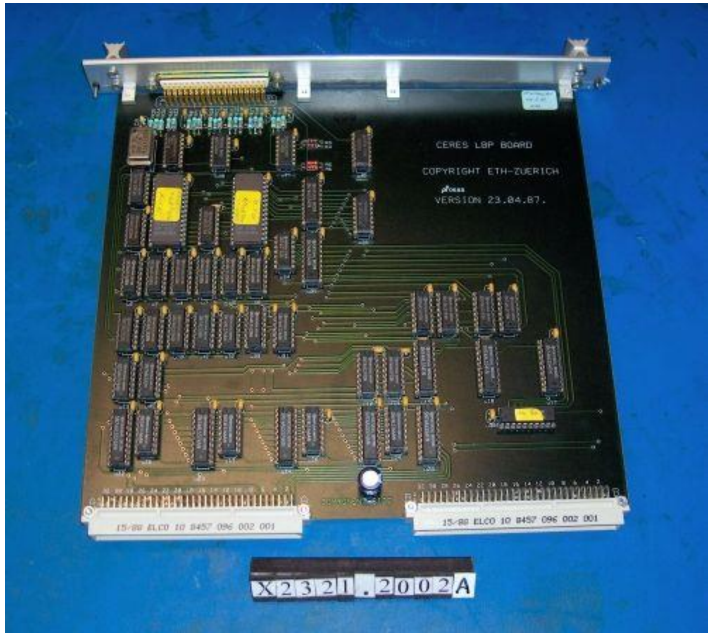
Its instruction set was optimized for / codesigned with the intermediate code of the Modula-2 Compiler.

It ran at 7 MHz and had a screen resolution of 704 x 927 pixels.



Some ETH History

1986: A 32-bit processor NS32032 CPU was used to build a new computer *Ceres* together with *its operating system Oberon* that was programmed using the *language Oberon*.

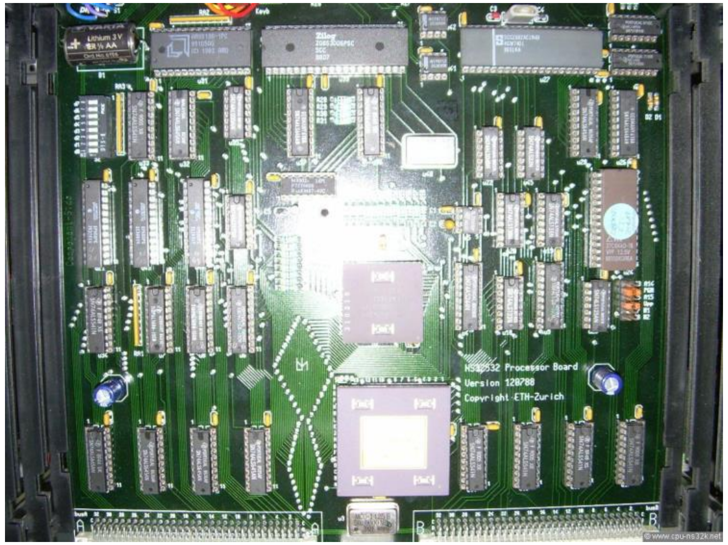


Sources: The Web Site to Remember National Semiconductor's Series 32000 Family, http://www.cpu-ns32k.net/Ceres.html

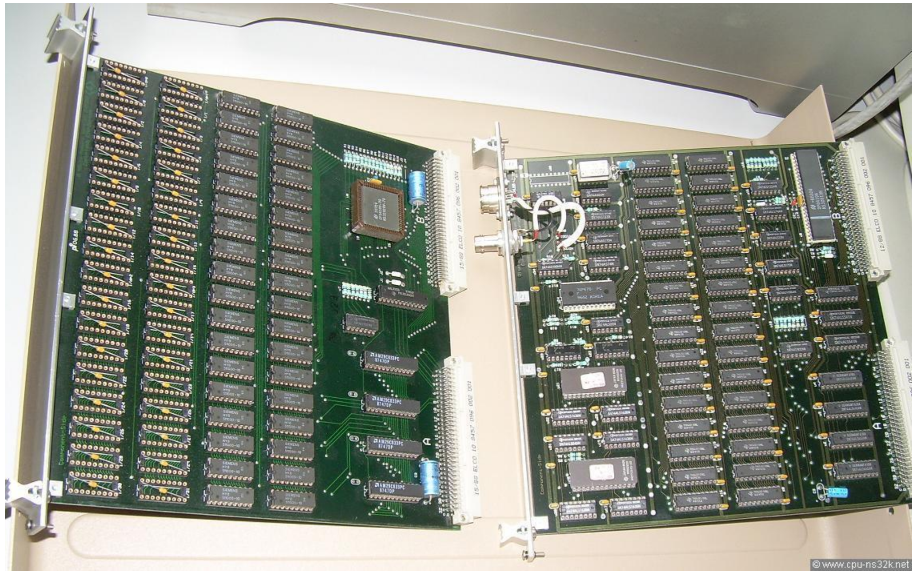
Some ETH History

1988 Ceres2, based on NS32532 CPU

cpu board, housing



cpu board



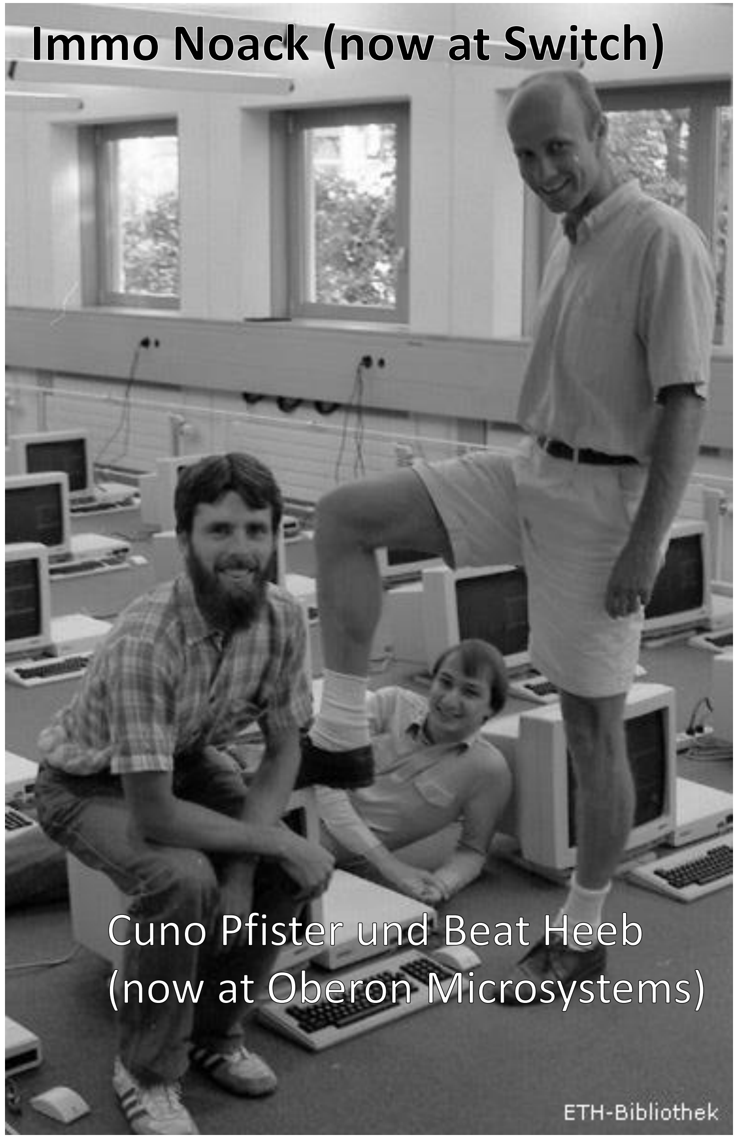
memory boards

Sources: The Web Site to Remember National Semiconductor's Series 32000 Family, http://www.cpu-ns32k.net/Ceres.html

Some ETH History

1991 Ceres 3, based on NS32GX32 CPU (cheaper, without MMU)

Used for education at ETH until 1999 (10s of machines)



Sources: The Web Site to Remember National Semiconductor's Series 32000 Family, http://www.cpu-ns32k.net/Ceres.html

Some ETH History

From mid 1990s

Oberon V4 availability as subsystems on Amiga, AtariST, DECStation, HP700, Linux, MacII, PowerMac, RS6000, SiliconGraphics, Solaris 2, Windows

System 3 available on Win3x, Win95NT, Unix (Darwin, PPC Linux, x86 Linux, x86 Solaris) , Macintosh (68k, PowerPC), with slim binaries

Native for various platforms.

From 2001: Aos / A2 (Active Oberon)

Background: Co-Design @ ETH

Languages (Pascal Family)

Modula → Oberon → Component Pascal

Oberon → ActiveOberon → (+MathOberon) → **Active Cells**

Oberon → Oberon07

ActiveOberon → Zonnon

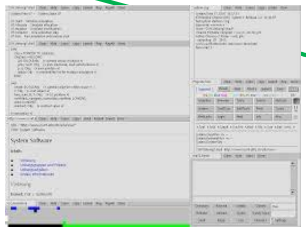
Operating / Runtime Systems

Medos → Oberon → BlackBox

Oberon → Aos → **A2** → **SoC**

A2 → **LockFree Kernel**

Oberon → HeliOs → **Minos**



Hardware

Lilith → Ceres → x86 / IA64/ ARM Emulations on Unix / Linux /MacOS → **TRM (FPGA)**

x86 / IA64/ ARM Emulations on Unix / Linux /MacOS → RISC (FPGA)

1980 1990 2000 2010

Course Overview

Part1: Contemporary Hardware

Case Study 1. Minos: Embedded System

- Safety-critical and fault-tolerant monitoring system
- Originally invented for autopilot system for helicopters
- Topics: ARM Architecture, Cross-Development, Object Files and Module Loading, Basic OS Core Tasks (IRQs, MMUs etc.), Minimal Single-Core OS: Scheduling, Device Drivers, Compilation and Runtime Support.
- With hands-on lab on Raspberry Pi (2)

Course Overview

Part1: Contemporary Hardware

Case Study 2. A2: A lock free Multiprocessor OS kernel

- Universal operating system for symmetric multiprocessors (SMP)
- Based on the co-design of a programming language (Active Oberon) and operating system kernel (A2)
- Topics: Intel SMP Architecture, Multicore Operating System, Scheduling, Synchronisation, Synchronous and Aysynchronous Context Switches, Priority Handling, Memory Handling, Garbage Collection.
- With hands-on labs on x86ish hardware and Raspberry Pi

Course Overview

Part2: Custom Designed Systems

Case Study 3. RISC: Single-Processor System [Lectures by Paul Reed]

- RISC single-processor system designed from scratch: hardware on FPGA
- Graphical workstation OS and compiler ("Project Oberon")
- Topics: building a system from scratch, Art of simplicity, Graphical OS, Processor Design.

Case Study 4. Active Cells: Multi-Processor System

- Special purpose heterogeneous system on a chip (SoC)
- Massively parallel hard- and software architecture based on Message Passing
- Topics: Dataflow-Computing, Tiny Register Machine: Processor Design Principles, Software-/Hardware Codesign, Hybrid Compilation, Hardware Synthesis

Organization

- Lecture Wednesday 13:15-15:00 (CAB H 52)
  with a break around 14:00

- Exercise Lab Wednsday 15:15 – 17:00 (CAB H 52)
  Guided, open lab, duration normally 2h
  First exercise: **today (September 25th)**

- Oral Examination in examination period after semester (15 minutes).
  Prerequisite: knowledge from both course and lab

Design Decisions: Area of Conflict

simple / undersized	Programming Model	sophisticated / complex
tailored / non-generic	Compiler	universal / overly generic
comprehensible / simplicistic	Language	elaborate / incomprehensible
customizable / inconvenient	Tools	feature rich / predetermined
economic / unoptimzed	System	optimized / uneconomic

I am about here

Minimal **O**perating **S**ystem

1. CASE STUDY MINOS

Topics

- Hardware platform
- Cross development
- Simple modular OS
- Runtime Support
- Realtime task scheduling
- I/O (SPI)*

*Serial Peripheral Interface,

Learn to Know the Target Architecture

1.1 HARDWARE

ARM Processor Architecture Family

- 32 bit **R**educed **I**nstruction **S**et **C**omputer architecture by ARM Holdings
  - 1st production 1985 (Acorn Risc Machine at 4MHz)
  - ARM Ltd. today does not sell hardware but (licenses and hardware descriptions for) chip designs
- Initial designs used for coprocessors in the 8-bit BBC Micro Computers
  (Computer Literacy Project in the 1980s)
- First ARM Computer: Archimedes (1987)
- An early prominent example: StrongARM (1995)
  - by DEC, licensing the design from **A**dvanced **R**isc **M**achines.
  - XScale implementation by Intel (now Marvell) after DEC take over
- More than 90 percent of the sold mobile phones (since 2007) contain at least one ARM processor (often more)*
  [95% of smart phones, 80% of digital cameras and 35% of all electronic devices*]
- Modular approach (today):
  ARM families produced for different profiles, such as Application Profile, Realtime Profile and Microcontroller / Low Cost Profile

BBC Micro

Acorn Archimedes

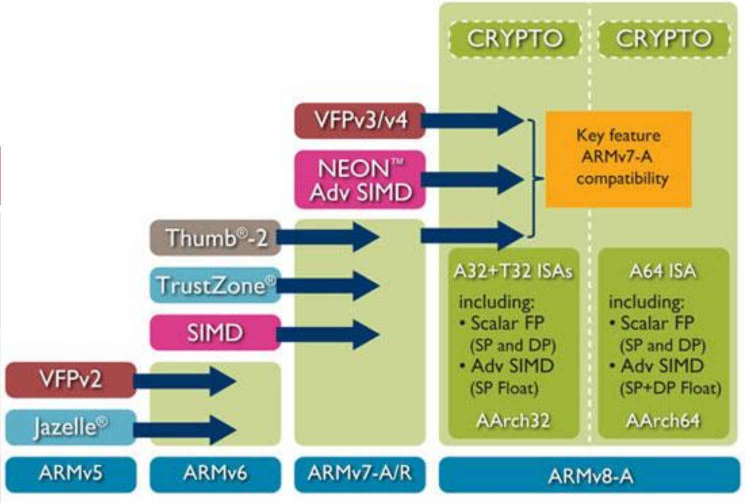
*http://news.cnet.com/ARMed-for-the-living-room/2100-1006_3-6056729.html
*http://arm.com/about/company-profile/index.php

Other Contemporary RISC Architectures

Examples

- MIPS (MIPS Technologies)
  - Business model similar to that of ARM
  - Architectures MIPS(I|…|V), MIPS(32|64), microMIPS(32|64)
- AVR (Atmel)
  - Initially targeted towards microcontrollers
  - Harvard Architecture designed and Implemented by Atmel
  - Families: tinyAVR, megaAVR, AVR32
  - AVR32: mixed 16-/32-bit encoding
- SPARC (Sun Microsystems)
  - Available as open-source: e.g. LEON (FPGA)
- MicroBlaze, PicoBlaze (Xilinx)
  - Softcore on FPGAs, support integrated in Linux.
- RISC-V (University of California, Berkeley)
  - Open Architecture, BSD-licensed

ARM *Architecture* Versions



[http://www.arm.com/products/processors/instruction-set-architectures/]

Architecture	Features
ARM v1-3	Cache from ARMv2a, 32-bit ISA in 26-bit address space
ARM v4	Pipeline, MMU, 32 bit ISA in 32 bit address space
ARM v4T	16-bit encoded Thumb Instruction Set
ARM v5TE	Enhanced DSP instructions, in particular for audio processing
ARM v5TEJ	Jazelle Technology extension to support Java acceleration technology (documentation restricted)
ARM v6	SIMD instructions, Thumb 2, Multicore, Fast Context Switch Extension
ARM v7	profiles: Cortex- A (applications), -R (real-time), -M (microcontroller)
ARM v8	Supports 64-bit data / addressing (registers). ARM 64 base instruction description: more than 500 of 6666 pages of the ARM Architecture Reference Manual

ARM Processor *Families (Microarchitectures)*

very much simplified & sparse

Architecture	Product Line / Family (Implementation)	Speed (MIPS)
ARMv1-ARMv3	ARM1-3, 6	4-28 (@8-33MHz)
ARMv3	ARM7	18-56 MHz
ARMv4T, ARMv5TEJ	ARM7TDMI	up to 60
ARMv4	StrongARM	up to 200 (@200MHz)
ARMv4	ARM8	up to 84 (@72MHz)
ARMv4T	ARM9TDMI	200 (@180MHz)
ARMv5TE(J)	ARM9E	220(@200MHz)
ARMv5TE(J)	ARM10E	
ARMv5TE	XScale	up to 1000 @1.25GHz
ARMv6	ARM11	740
ARMv6, ARMv7, ARMv8	ARM Cortex	up to 10000 DMIPS (Multicore @2GHz)

ARM Cortex Microarchitectures

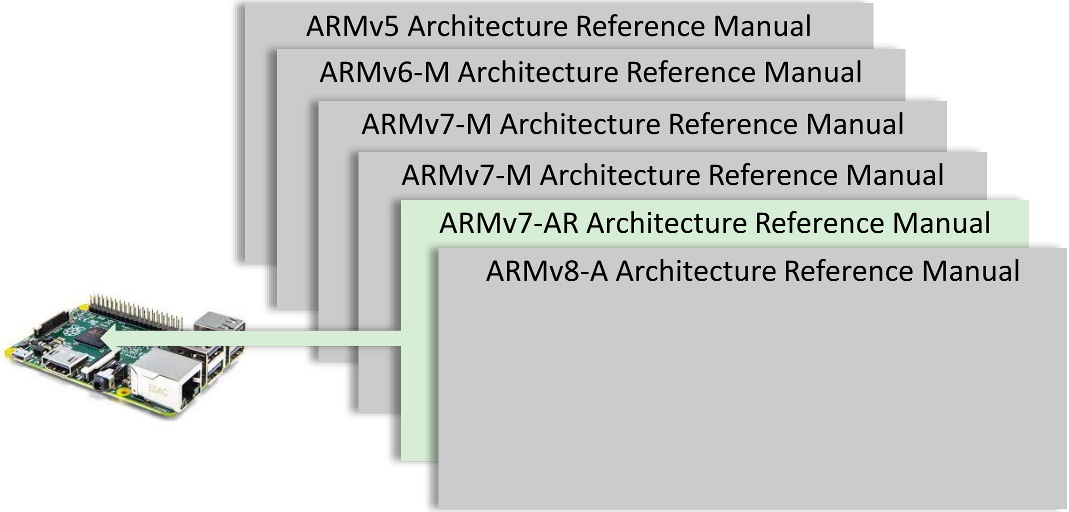
- Cortex-A
  - ARM v7-A, ARM v8-A
  - Application profile: typically including luxuries such as MMU support for OSes, ranging up to high performance multicore CPUs with (NEON) SIMD units while power consumption is moderate, newest generation provides 64-bit support
- Cortex-M
  - ARM v6-M, ARM v7-M
  - Microcontroller profile (32bit), Thumb instruction set, very low power consumption, some provide a MPU
- Cortex-R
  - ARM v7-R
  - Realtime profile, tightly coupled memory, deterministic interrupt handling, redundant computation (HW replication for fault tolerance)

cf. https://en.wikipedia.org/wiki/List_of_ARM_microarchitectures

ARM Architecture Reference Manuals

describe

- ARM/Thumb instruction sets
- Processor modes and states
- Exception and interrupt model
- System programmer's model, standard coprocessor interface



- Memory model, memory ordering and memory management for different potential implementations
- Optional extensions like Floating Point, SIMD, Security, Virtualization ...

for example required for the implementation of assembler, disassembler, compiler, linker and debugger and for the systems programmer.

ARM Technical System Reference Manuals

describe

- Particular processor implementation of an ARM architecture
- Redundant information from the Architecture manual (e.g. system control processor)
- Additional processor implementation specifics e.g. cache sizes and cache handling, interrupt controller, generic timer

usually required by a system's programmer

Cortex™-A7 MPCore™
Technical Reference Manual

System on Chip Implementation Manuals

describe

- Particular implementation of a System on Chip
- Address map:
  physical addresses and
  bit layout for the registers
- Peripheral components / controllers,
  such as Timers, Interrupt controller, GPIO, USB, SPI, DMA, PWM, UARTs

usually required by a system's programmer.

BCM2835 ARM Peripherals

ARM Instruction Set

consists of

- Data processing instructions
- Branch instructions
- Status register transfer instructions
- **Load and Store** instructions
- Generic Coprocessor instructions
- Exception generating instructions

Some Features

of the ARM Instruction Set

- 32 bit instructions / many in one cycle / 3 operands

- Load / store architecture (no memory operands such as in x86)

```
ldr r11, [fp, #-8]
add r11, r11, #1
str r11, [fp, #-8]
```

?

Some Features

of the ARM Instruction Set

- 32 bit instructions / many in one cycle / 3 operands

- Load / store architecture (no memory operands such as in x86)

```
ldr r11, [fp, #-8]
add r11, r11, #1
str r11, [fp, #-8]
```

increment a
local variable

Some Features

of the ARM Instruction Set

- Index optimized instructions (such as pre-/post-indexed addressing)

```
stmdb sp!,{fp,lr} ; store multiple decrease before and update sp

    …                                  ?

ldmia sp!,{fp,pc} ; load multiple increase after and update sp
```

Some Features

of the ARM Instruction Set

- Index optimized instructions (such as pre-/post-indexed addressing)

```
stmdb sp!,{fp,lr} ; store multiple decrease before and update sp
```

…

stack activation frame

```
ldmia sp!,{fp,pc} ; load multiple increase after and update sp
```

Some Features

of the ARM Instruction Set

- *Predication*: all instructions can be conditionally executed*

```
cmp   r0, #0
swieq #0xa
```

?

Some Features

of the ARM Instruction Set

- *Predication*: all instructions can be conditionally executed*

```
cmp   r0, #0
swieq #0xa
```

null pointer check

Impressive Example of Predication

```
loop:  CMP    Ri, Rj        ; set condition flags
       SUBGT  Ri, Ri, Rj    ; if i>j then i = i-j;
       SUBLT  Rj, Rj, Ri    ; if i<j then j = j-i;
       BNE  loop            ; if i != j then loop
```

Some Features

of the ARM Instruction Set

Link Register

bl #0x0a0100070 ?

- Shift and rotate in instructions

add r11, fp, r11, lsl #2 ?

Some Features

of the ARM Instruction Set

Link Register

bl #0x0a0100070

procedure call

- Shift and rotate in instructions

add r11, fp, r11, lsl #2

r11 = fp + r11*4
e.g. array access

Some Features

of the ARM Instruction Set

- PC-relative addressing

```
ldr r0, [pc, #+24]
```

?

- Coprocessor access instructions

```
mrc p15, 0, r11, c6, c0, 0
```

?

Some Features

of the ARM Instruction Set

- PC-relative addressing

ldr r0, [pc, #+24]

load a large constant

- Coprocessor access instructions

mrc p15, 0, r11, c6, c0, 0

setup the mmu

ARM Instruction Set

Encoding (ARM v5)

Instruction	Encoding (bits 31 … 0)
Data processing immediate shift	cond [1] | 0 0 0 | opcode | S | Rn | Rd | shift amount | shift | 0 | Rm
Miscellaneous instructions: See Figure 3-3	cond [1] | 0 0 0 | 1 0 x x | 0 | x x x x x x x x x x x x x x x | 0 | x x x x
Data processing register shift [2]	cond [1] | 0 0 0 | opcode | S | Rn | Rd | Rs | 0 | shift | 1 | Rm
Miscellaneous instructions: See Figure 3-3	cond [1] | 0 0 0 | 1 0 x x | 0 | x x x x x x x x x x x x | 0 | x x | 1 | x x x x
Multiplies, extra load/stores: See Figure 3-2	cond [1] | 0 0 0 | x x x x x | x x x x x x x x x x x x | 1 | x x | 1 | x x x x
Data processing immediate [2]	cond [1] | 0 0 1 | opcode | S | Rn | Rd | rotate | immediate
Undefined instruction [3]	cond [1] | 0 0 1 | 1 0 | x | 0 0 | x x x x x x x x x x x x x x x x x x x x
Move immediate to status register	cond [1] | 0 0 1 | 1 0 | R | 1 0 | Mask | SBO | rotate | immediate
Load/store immediate offset	cond [1] | 0 1 0 | P | U | B | W | L | Rn | Rd | immediate
Load/store register offset	cond [1] | 0 1 1 | P | U | B | W | L | Rn | Rd | shift amount | shift | 0 | Rm
Undefined instruction	cond [1] | 0 1 1 | x x x x x x x x x x x x x x x x x x x x | 1 | x x x x
Undefined instruction [4,7]	1 1 1 1 | 0 | x x x x x x x x x x x x x x x x x x x x x x x x x x x
Load/store multiple	cond [1] | 1 0 0 | P | U | S | W | L | Rn | register list
Undefined instruction [4]	1 1 1 1 | 1 0 0 | x x x x x x x x x x x x x x x x x x x x x x x x x
Branch and branch with link	cond [1] | 1 0 1 | L | 24-bit offset
Branch and branch with link and change to Thumb [4]	1 1 1 1 | 1 0 1 | H | 24-bit offset
Coprocessor load/store and double register transfers [6]	cond [5] | 1 1 0 | P | U | N | W | L | Rn | CRd | cp_num | 8-bit offset
Coprocessor data processing	cond [5] | 1 1 1 0 | opcode1 | CRn | CRd | cp_num | opcode2 | 0 | CRm
Coprocessor register transfers	cond [5] | 1 1 1 0 | opcode1 | L | CRn | Rd | cp_num | opcode2 | 1 | CRm
Software interrupt	cond [1] | 1 1 1 1 | swi number
Undefined instruction [4]	1 1 1 1 | 1 1 1 1 | x x x x x x x x x x x x x x x x x x x x x x x x

shiftable register

conditional execution

8 bit immediates with even rotate

load / store with destination increment

undefined instruction: user extensibility

load / store with multiple registers

branches with 24 bit offset

generic coprocessor instructions

From ARM Architecture Reference Manual

Thumb Instruction Set

ARM instruction set complemented by

- Thumb Instruction Set
  - 16-bit instructions, 2 operands
  - eight GP registers accessible from most instructions
  - subset in functionality of ARM instruction set
  - targeted for density from C-code (~65% of ARM code size)

- Thumb2 Instruction Set
  - extension of Thumb, adds 32 bit instructions to support almost all of ARM ISA (different from ARM instruction set encoding!)
  - design objective: ARM performance with Thumb density

Typical procedure call on ARM

Caller: push parameters ...

use branch and link instruction. Stores the PC of the next instruction into the link register. bl #address

Callee: save link register and frame pointer on stack and set new frame pointer.

```
stmdb sp!, {fp, lr}
mov fp, sp
```

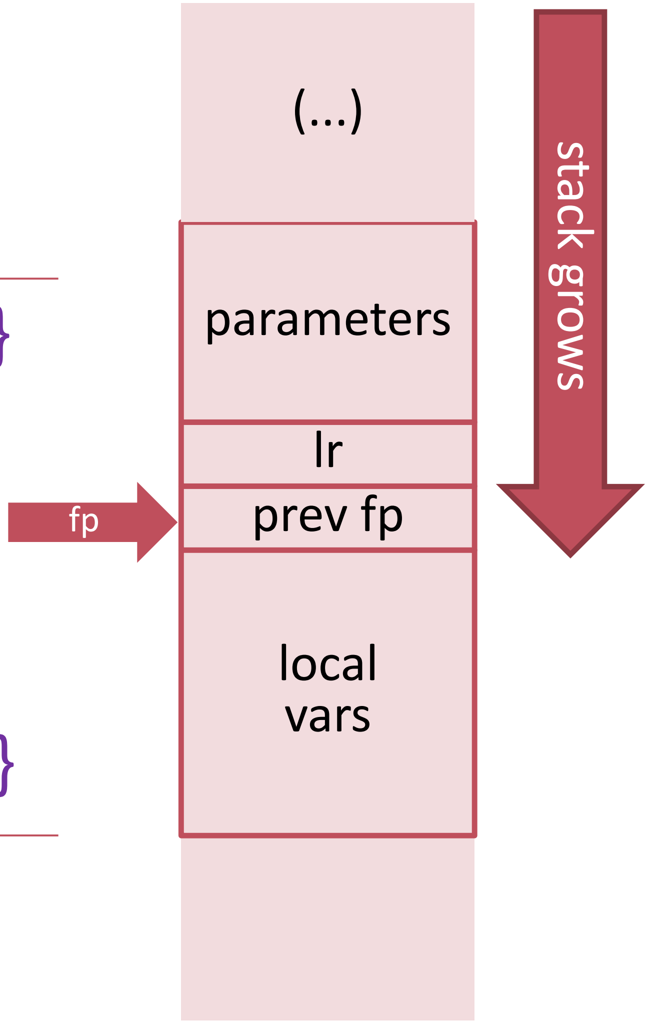
Execute procedure content ...

Reset stack pointer and restore frame pointer and and jump back to caller address.

```
mov sp, fp
ldmia sp!, {fp, pc}
```

Caller: cleanup parameters from stack

```
add sp, sp, #n
...
```



ARM Processor Modes

ARM from v5 has (at least) seven basic operating modes

- Each mode has access to its **own stack** and a different subset of registers
- Some operations can only be carried out in a privileged mode

Mode	Description / Cause
Supervisor	Reset / Software Interrupt
FIQ	Fast Interrupt
IRQ	Normal Interrupt
Abort	Memory Access Violation
Undef	Undefined Instruction
System	Privileged Mode with same registers as in User Mode
User	Regular Application Mode

privileged

exceptions

normal execution

ARM Register Set

ARM has 37 registers, all 32-bits long
A subset is accessible in each mode
Register 13 is the Stack Pointer (by convention)
Register 14 is the Link Register**
Register 15 is the Program Counter (settable)
CPSR* is not immediately accessible

Shadowing

	User/System	FIQ	IRQ	SVC	UND	ABT
unbanked	R0					
	R1					
	R2					
	R3					
	R4					
	R5					
	R6					
	R7					
banked	R8	R8.FIQ				
	R9	R9.FIQ				
	R10	R10.FIQ				
	R11	R11.FIQ				
	R12	R12.FIQ				
	R13 SP	R13.FIQ SP	R13.IRQ SP	R13.SVC SP	R13.UND SP	R13.ABT SP
	R14 LR	R14.FIQ LR	R14.IRQ LR	R14.SVC LR	R14.UND LR	R14.ABT LR
	R15 PC					
	CPSR*					
		SPSR*.FIQ	SPSR.IRQ	SPSR.SVC	SPSR.UND	SPSR.ABT

* current / saved processor status register, accessible via MSR / MRS instructions
** more than a convention: link register set as side effect of some instructions

Processor Status Register (PSR)

Condition Codes

- N=**N**egative result from ALU
- Z=**Z**ero result from ALU
- C=ALU operation **C**arried out *
- V=ALU operation o**v**erflowed

Interrupt Disable bits

- I=1: Disables IRQ
- F=1: Disables FIQ

Mode Bits

- Specify processor mode

Other bits

- architecture 5TE(J) and later
  - Q flag: sticky overflow flag for saturating instr.
  - J flag: Jazelle state
- architecture 6 and later
  - GE[0:3]: used by SIMD instructions
  - E: controls endianess
  - A: controls imprecise data aborts
  - IT: controls conditional execution of Thumb2

T Bit

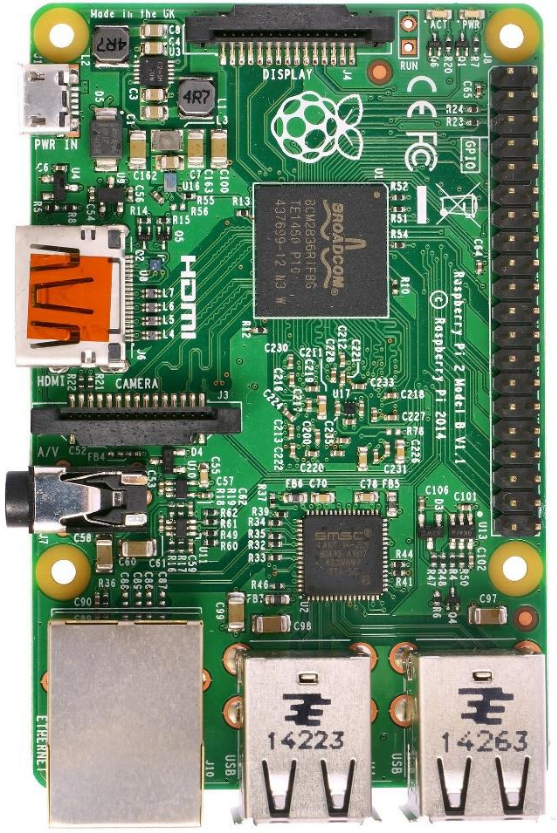
- T=0: Processor in ARM mode
- T=1: Processor in Thumb State
- Introduced in Architecture 4T

* reverse cmp/sub meaning compared with x86

Raspberry Pi 2

Raspberry Pi 2 (Model B) will be the hardware used at least in the first 4 weeks lab sessions

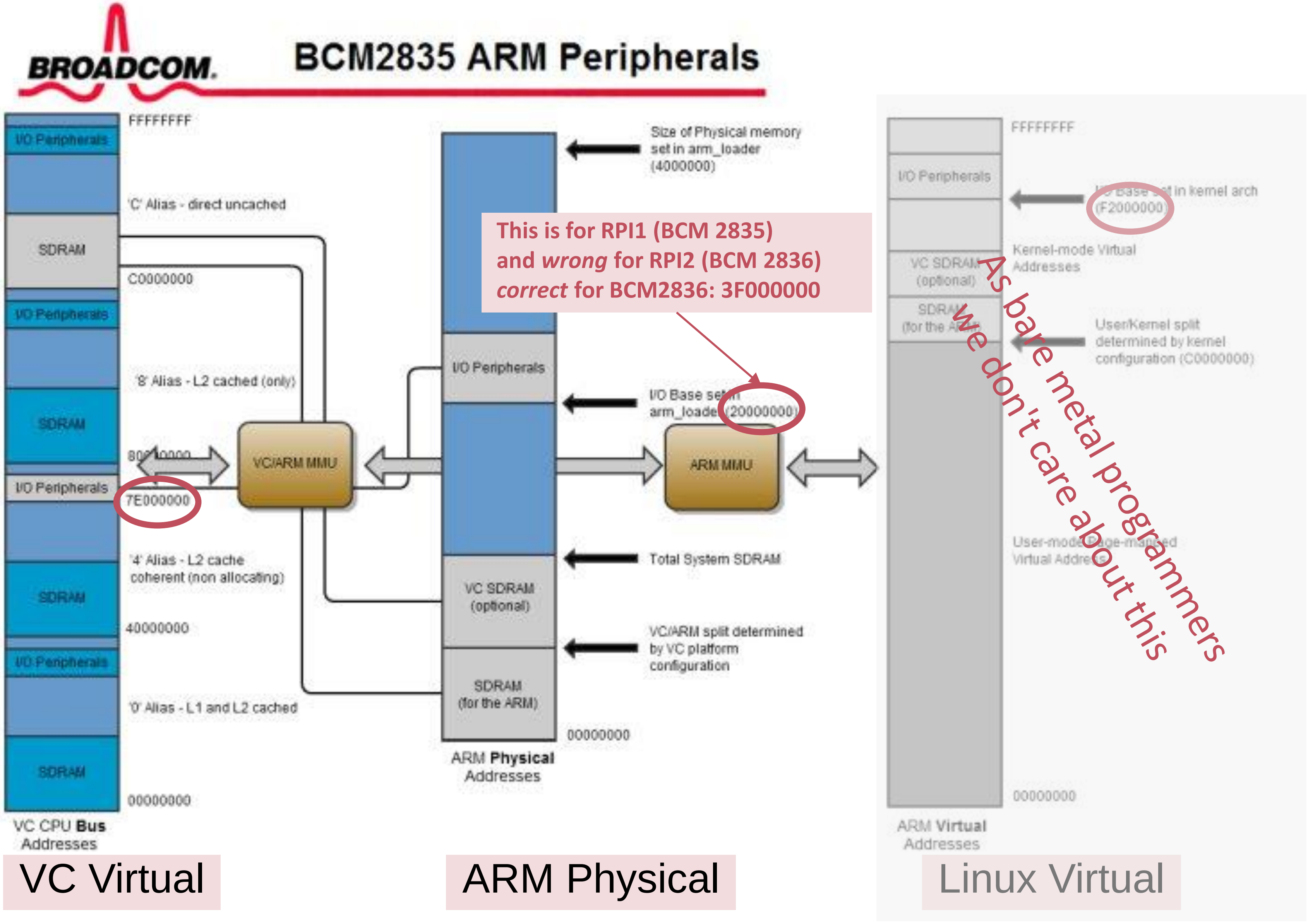
- Produced by element14 in the UK (www.element14.com)
- Features
  - Broadcom BCM2836 ARMv7
    Quad Core Processor running at 900 MHz
  - 1G RAM
  - 40 PIN GPIO
  - Separate GPU ("Videocore")
  - Peripherals: UART, SPI, USB, 10/100 Ethernet Port (via USB), 4pin Stereo Audio, CSI camera, DSI display, Micro SD Slot
  - Powered from Micro USB port



ARM System Boot

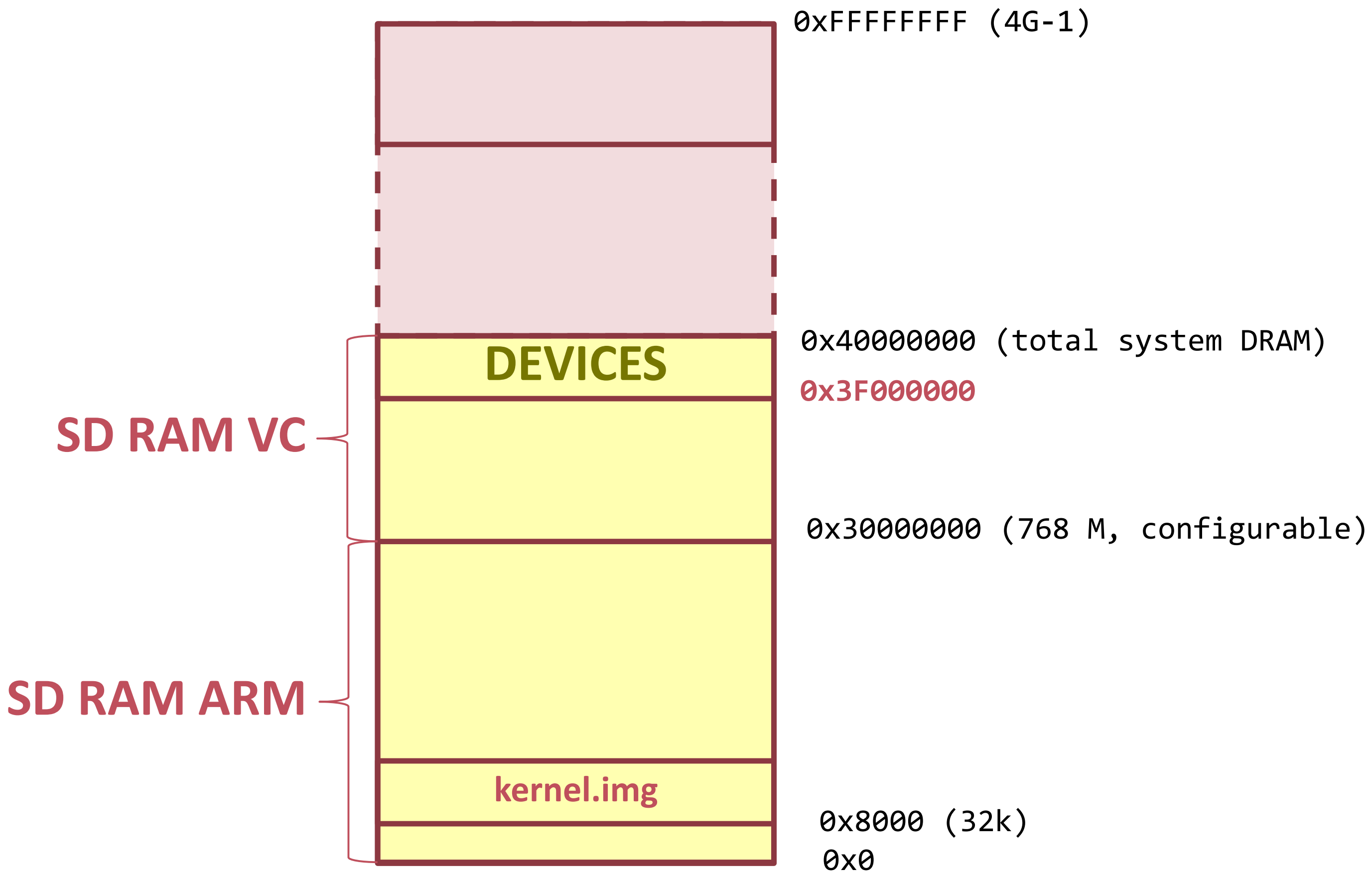
- ARM processors usually starts executing code at adr 0x0
  - e.g. containing a branch instruction to jump over the interrupt vectors
  - usually requires some initial setup of the hardware
- The RPI, however, is **booted from the Video Core CPU** (VC):
  the firmware of the RPI does a lot of things before we get control:
  kernel-image gets copied to address 0x8000H and branches there
  No virtual to physical address-translation takes place in the start.
- Only one core runs at that time. (More on this later)

RPI 1 Memory Map



RPI 2 Memory Map

- Initially the MMU is switched off. No memory translation takes place.
- System memory divided in ARM and VC part, partially shared (e.g. frame buffer)
- ARM's memory mapped registers start from **0x3F000000** -- opposed to reported offset ~~0x7E000000~~ in BCM 2835 Manual



General Purpose I/O (GPIO)

- Software controlled processor pins
  - Configurable direction of transfer
  - Configurable connection
    - → with internal controller (SPI, MMC, memory controller, …)
    - → with external device
- Pin state settable & gettable
  - High, low
- Forced interrupt on state change
  - On falling/ rising edge

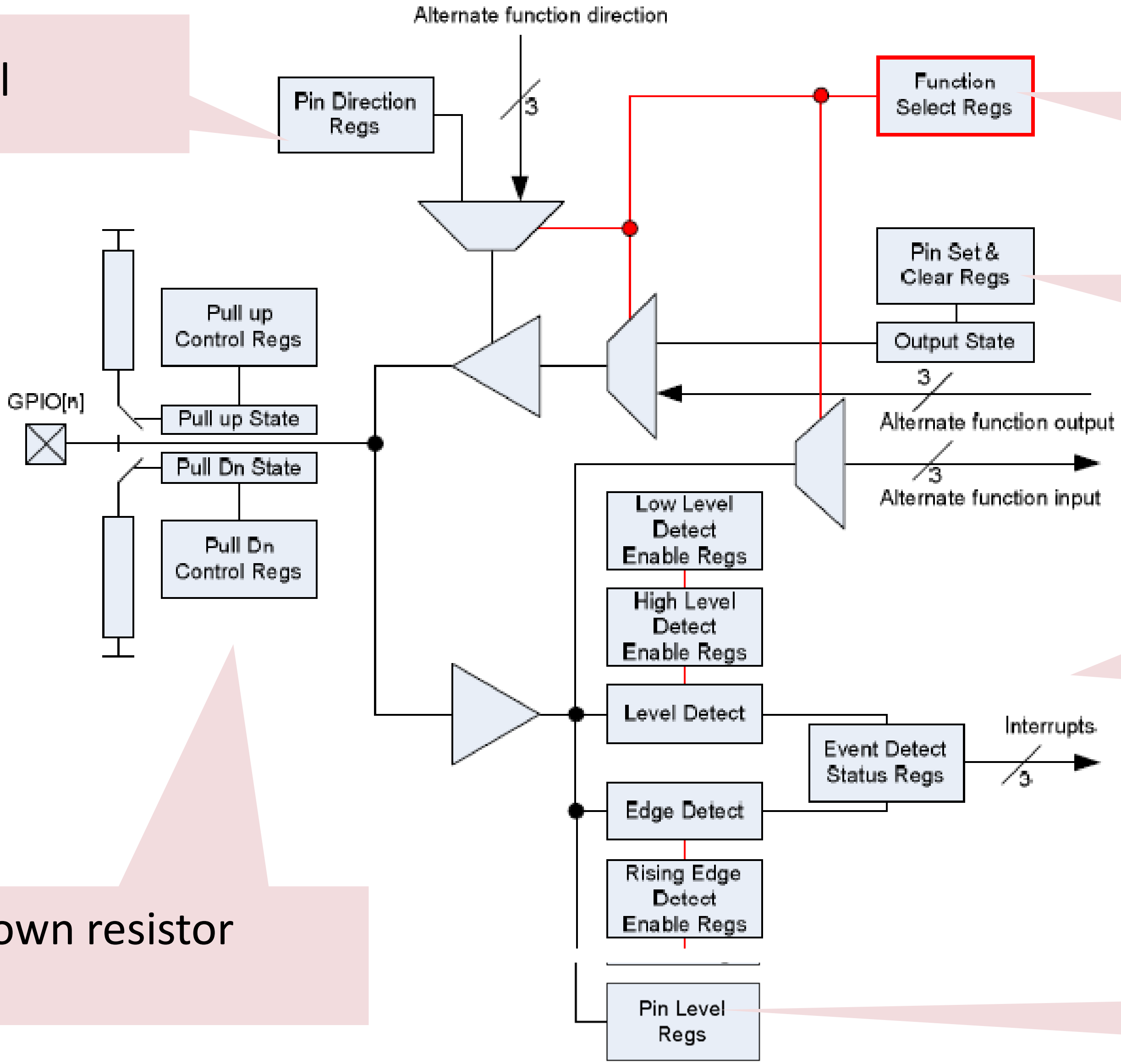
GPIO

Block Diagram (BCM 2835)

pin direction control

internal function selection

output control registers



interrupt control

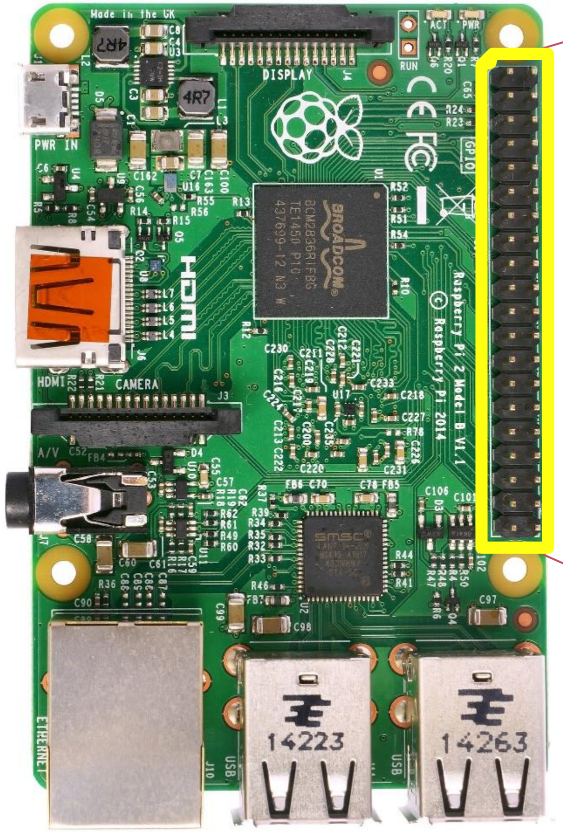
pull up / down resistor control

input (pin level) registers

Raspberry Pi 2 GPIO Pinout

Connecting external power with 5 v here kills the board!

Be careful with the USB TTL Cable (Exercise 2)



name	pin	pin	name
3.3 V DC	01	02	DC power 5v
GPIO 02	03	04	DC power 5v
GPIO 03	05	06	ground
GPIO 04	07	08	GPIO 14
ground	09	10	GPIO 15
GPIO 17	11	12	GPIO 18
GPIO 27	13	14	ground
GPIO 22	15	16	GPIO 23
3.3V DC	17	18	GPIO 24
GPIO 10	19	20	ground
GPIO 09	21	22	GPIO 25
GPIO 11	23	24	GPIO 08
ground	25	26	GPIO 07
ID_SD	27	28	ID_SC
GPIO 05	29	30	ground
GPIO 06	31	32	GPIO 12
GPIO 13	33	34	ground
GPIO 19	35	36	GPIO 16
GPIO 26	37	38	GPIO 20
ground	39	40	GPIO 21

Documentation Examples (BCM2835 ARM Peripherals)

GPIO Register Overview (p. 90)

Address	Field Name	Description	Size	Read/ Write
~~0x 7E20 0000~~	~~GPFSEL0~~	~~GPIO Function Select 0~~	~~32~~	~~R/W~~
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0004	GPFSEL1	GPIO Function Select 1	32	R/W
0x 7E20 0008	GPFSEL2	GPIO Function Select 2	32	R/W
0x 7E20 000C	GPFSEL3	GPIO Function Select 3	32	R/W
0x 7E20 0010	GPFSEL4	GPIO Function Select 4	32	R/W

GPIO Pin Mapping / Alternate Functions (p.102)

GPIO13	Low	PWM1	SD5	<reserved>			ARM_TCK
GPIO14	Low	TXD0	SD6	<reserved>			TXD1
GPIO15	Low	RXD0	SD7	<reserved>			RXD1
GPIO16	Low	<reserved>	SD8	<reserved>	CTS0	SPI1_CE2_N	CTS1
GPIO17	Low	<reserved>	SD9	<reserved>	RTS0	SPI1_CE1_N	RTS1

GPIO Function Select (p. 92 -94)

Bit(s)	Field Name	Description	Type	Reset
31-30	---	Reserved	R	0
29-27	FSEL19	FSEL19 - Function Select 19 000 = GPIO Pin 19 is an input 001 = GPIO Pin 19 is an output 100 = GPIO Pin 19 takes alternate function 0 101 = GPIO Pin 19 takes alternate function 1 110 = GPIO Pin 19 takes alternate function 2 111 = GPIO Pin 19 takes alternate function 3 011 = GPIO Pin 19 takes alternate function 4 010 = GPIO Pin 19 takes alternate function 5	R/W	0
26-24	FSEL18	FSEL18 - Function Select 18	R/W	0
23-21	FSEL17	FSEL17 - Function Select 17	R/W	0
20-18	FSEL16	FSEL16 - Function Select 16	R/W	0
17-15	FSEL15	FSEL15 - Function Select 15	R/W	0
14-12	FSEL14	FSEL14 - Function Select 14	R/W	0
11-9	FSEL13	FSEL13 - Function Select 13	R/W	0
8-6	FSEL12	FSEL12 - Function Select 12	R/W	0
5-3	FSEL11	FSEL11 - Function Select 11	R/W	0
2-0	FSEL10	FSEL10 - Function Select 10	R/W	0

Table 6-3 – GPIO Alternate function select register 1

GPIO Setup (RPI2)

1. Program GPIO Pin Function (in / out / alternate function)
   by writing corresponding (memory mapped) GPFSEL register.
   GPFSELn: pins $10n, \ldots, 10n + 9$
   Use RMW (Read-Modify-Write) operation in order to keep the other bits

2. Use GPIO Pin

   a. If writing: set corresponding bit in the GPSETn or GPCLRn register
      set pin: **GPSETn**: pins $32n, \ldots, 32n + 31$
      clear pin: **GPCLRn**: pins $32n, \ldots, 32n + 31$
      no RMW (Read – Modify – Write) required.

   b. If reading: read corrsponding bit in the GPLEVn register
      GPLEVn: pins$32n, \ldots, 32n + 31$

   c. If "alternate function": device acts autonomously. Implement device driver.