ARM architecture

ARM7, ARM9, TDMI...

Brief history of ARM

- ARM is short for Advanced Risc Machines Ltd.
- Founded 1990, owned by Acorn, Apple and VLSI
- Known before becoming ARM as computer manufacturer Acorn which developed a 32-bit RISC processor for it's own use (used in Acorn Archimedes)

Why ARM here?

- ARM is one of the most licensed and thus widespread processor cores in the world
- Used especially in portable devices due to low power consumption and reasonable performance (MIPS / watt)
- Several interesting extensions available or in development like Thumb instruction set and Jazelle Java machine

ARM

- Processor cores: ARM6, ARM7, ARM9, ARM10, ARM11
- Extensions: Thumb, El Segundo, Jazelle etc.
- IP-blocks: UART, GPIO, memory controllers, etc

CPU	Description	ISA	Process	Voltage	Area mm2	Power mW	Clock / MHz	Mips / MHz
ARM7TDMI	Core	V4T	0.18u	1.8V	0.53	<0.25	60-110	0.9
ARM7TDMI-S	Synthesizable core	V4T	0.18u	1.8V	<0.8	<0.4	>50	0.9
ARM9TDMI	Core	V4T	0.18u	1.8V	1.1	0.3	167-220	1.1
ARM920T	Macrocell 16+16kB cache	V4T	0.18u	1.8V	11.8	0.9	140-200	1.05
ARM940T	Macrocell 8+8kB cache	V4T	0.18u	1.8V	4.2	0.85	140-170	1.05
ARM9E-S	Synthesizable core	V5TE	0.18u	1.8V	?	~1	133-200	1.1
ARM1020E	Macrocell 32+32kB cache	V5TE	0.15u	1.8V	~10	~0.85	200-400	1.25

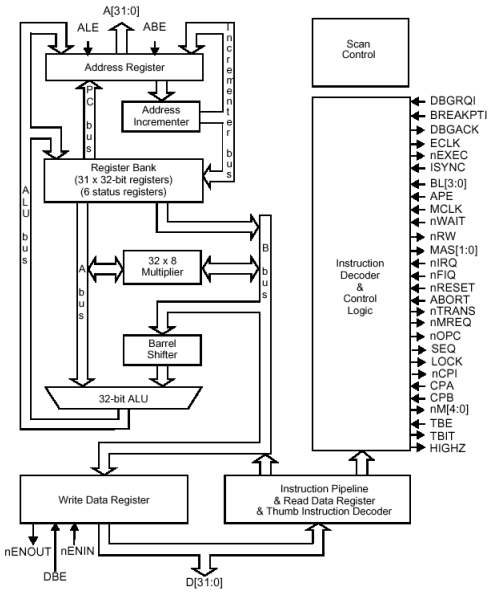
ARM architecture

- ARM:
- 32-bit RISC-processor core (32-bit instructions)
- 37 pieces of 32-bit integer registers (16 available)
- Pipelined (ARM7: 3 stages)
- Cached (depending on the implementation)
- Von Neuman-type bus structure (ARM7), Harvard (ARM9)
- 8 / 16 / 32 -bit data types
- 7 modes of operation (usr, fiq, irq, svc, abt, sys, und)

- Simple structure -> reasonably good speed / power consumption ratio

ARM7 internals

- Core block diagram:

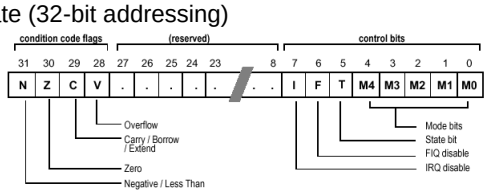


ARM7 internals

- ARM core modes of operation:
  - User (usr): Normal program execution state
  - FIQ (fiq): Data transfer state (fast irq, DMA-type transfer)
  - IRQ (iqr): Used for general interrupt services
  - Supervisor (svc): Protected mode for operating system support
  - Abort mode (abt): Selected when data or instruction fetch is aborted
  - System (sys): Operating system ‘privilege’-mode for user
  - Undefined (und): Selected when undefined instruction is fetched

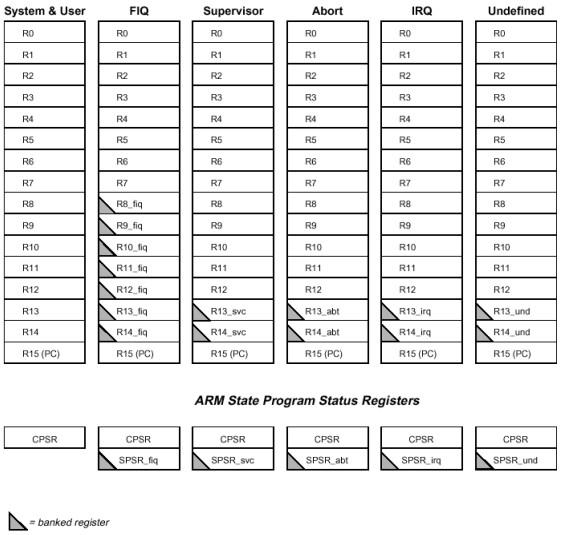
ARM7 register set

- Register structure depends on mode of operation
- 16 pieces of 32-bit integer registers R0 - R15 are available in ARM-mode (usr, user)
- R0 - R12 are general purpose registers
- R13 is Stack Pointer (SP)
- R14 is subroutine Link Register
  - Holds the value of R15 when BL-instruction is executed
- R15 is Program Counter (PC)
  - Bits 1 and 0 are zeroes in ARM-state (32-bit addressing)
- R16 is state register (CPSR, Current Program Status Regist



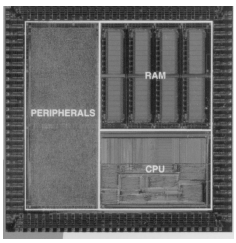
ARM7 register set

- There are 37 ARM registers in total of which variable amount is available as banked registers depending on the mode of operation
- R13 functions always as stack pointer
- R14 functions as link register in other than sys and usr - modes
- SPSR = Saved Program Status Register
- Flag register Mode-bits tell the processor operating mode and thus the registers available



ARM7TDMI

- TDMI = (?)
  - Thumb instruction set
  - Debug-interface (JTAG/ICEBreaker)
  - Multiplier (hardware)
  - Interrupt (fast interrupts)
- The most used ARM-version

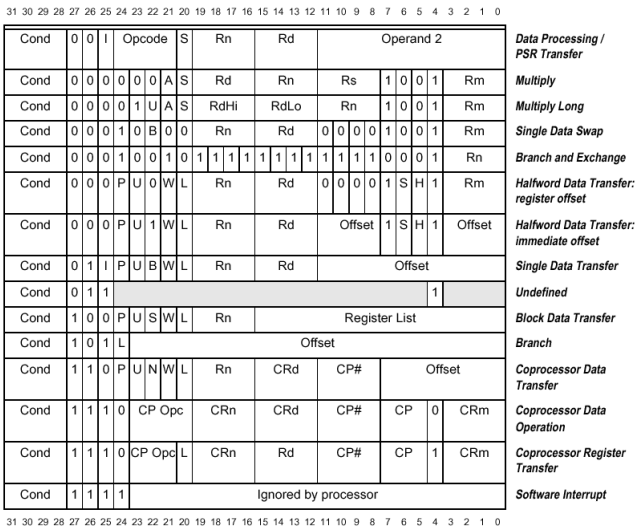


ARM instruction set

- Fully 32-bit instruction set in native operating mode
  - 32-bit long instruction word
- All instructions are conditional
  - Normal execution with condition AL (always)
- For a RISC-processor, the instruction set is quite diverse with different addressing modes

ARM instruction set

- Instruction word length 32-bits
- 36 instruction formats



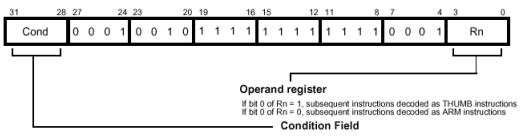
ARM instruction set

- All instructions are conditional
- In normal instruction execution (unconditional) condition field contents of AL is used (Always)
- In conditional operations one of the 14 available conditions is selected
- For example, instruction known usually as BNZ in ARM is NE (Z-flag clear) conditioned branch-instruction

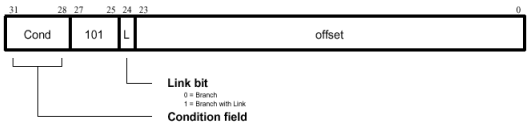
Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

Branching

- BX, Branch and eXchange
- Branch with instruction set exchange (ARM <-> Thumb)

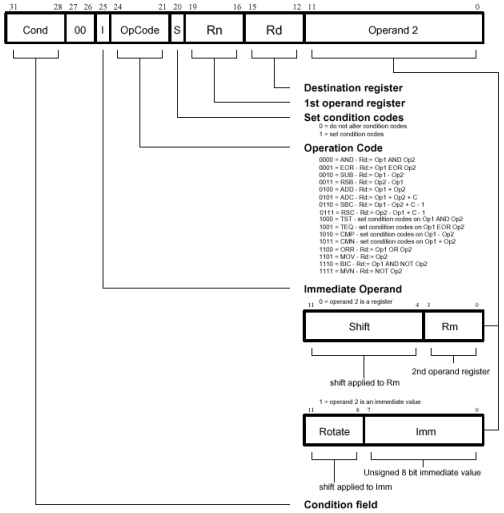


- B and BL
- Branch with 24-bit signed offset
- Link: PC -> R14



Data processing

- AND, EOR, SUB, RSB, ADD, ADC, SBC, RSC, TST, TEQ, CMP, CMN, ORR, MOV, BIC, MVN
- Multiple operation instruction

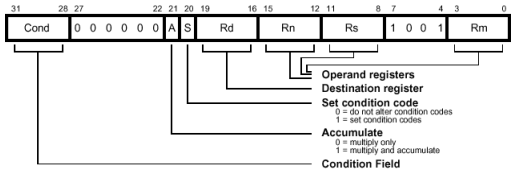


Data processing

Assembler Mnemonic	OpCode	Action
AND	0000	operand1 AND operand2
EOR	0001	operand1 EOR operand2
SUB	0010	operand1 - operand2
RSB	0011	operand2 - operand1
ADD	0100	operand1 + operand2
ADC	0101	operand1 + operand2 + carry
SBC	0110	operand1 - operand2 + carry - 1
RSC	0111	operand2 - operand1 + carry - 1
TST	1000	as AND, but result is not written
TEQ	1001	as EOR, but result is not written
CMP	1010	as SUB, but result is not written
CMN	1011	as ADD, but result is not written
ORR	1100	operand1 OR operand2
MOV	1101	operand2 (operand1 is ignored)
BIC	1110	operand1 AND NOT operand2 (Bit clear)
MVN	1111	NOT operand2 (operand1 is ignored)

Multiplication

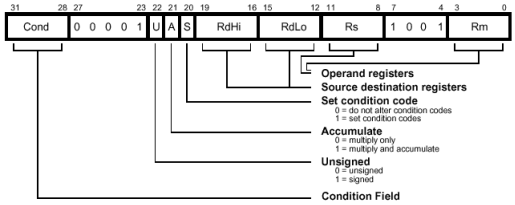
- MUL, MLA



Examples

```
MUL             R1,R2,R3 ; R1:=R2*R3
MLAEQS          R1,R2,R3,R4; Conditionally R1:=R2*R3+R4,
                         ; setting condition codes.
```

- MULL, MLAL



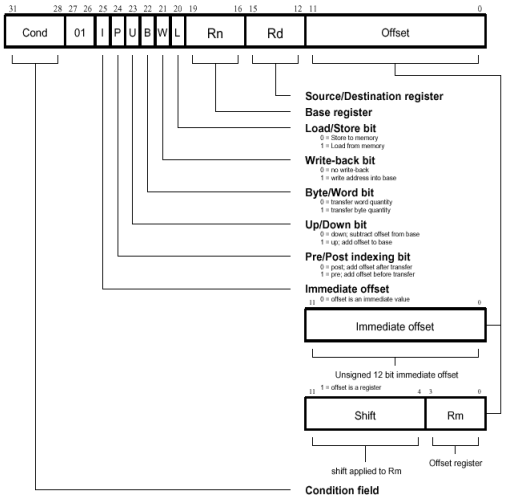
Examples

```
UMULL           R1,R4,R2,R3; R4,R1:=R2*R3
UMLALS          R1,R5,R2,R3; R5,R1:=R2*R3+R5,R1 also setting
                         ; condition codes
```

Mnemonic	Description	Purpose
UMULL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply Long	32 x 32 = 64
UMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply & Accumulate Long	32 x 32 + 64 = 64
SMULL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply Long	32 x 32 = 64
SMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply & Accumulate Long	32 x 32 + 64 = 64

Data transfer

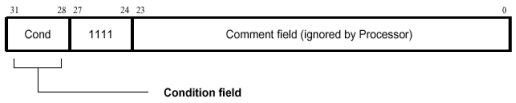
- LDR, STR



- Other data transfer operations: LDRH, STRH, LDRSB, LDRSH, LDM, STM, SWP

Exception

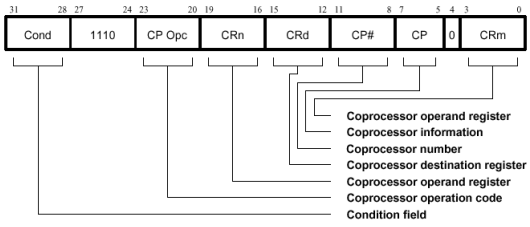
- SWI: SoftWare Interrupt
- Transfers execution to address in memory location 0x8 and changes the mode to svc.
- Comment field allows the interrupt service to determine the wanted action for SWI.



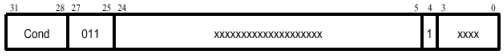
Other instructions

- Coprocessor instructions: CDP, LDC, STC, MRC, MCR
- ARM does not execute these instructions but lets a coprocessor to handle them

CDP:



Undefined instruction:

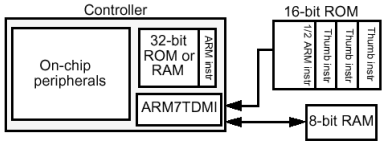


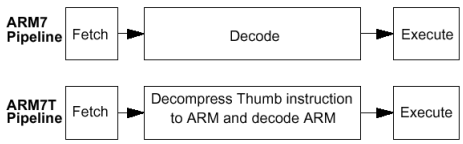
ARM Thumb

“Peukalo” ARM...

ARM Thumb

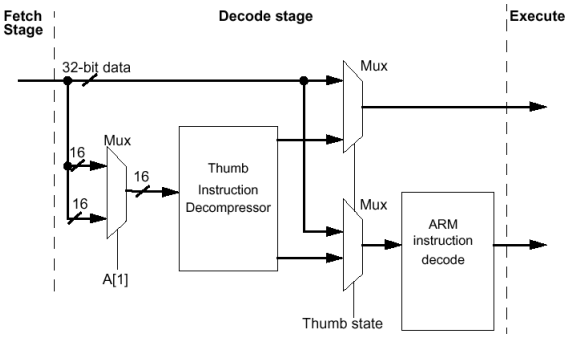
- T (Thumb)-extension shrinks the ARM instruction set to 16-bit word length -> 35-40% saving in amount of memory compared to 32-bit instruction set
- Extension enables simpler and significantly cheaper realization of processor system. Instructions take only half of memory than with 32-bit instruction set without significant decrease in performance or increase in code size.
- Extension is made to instruction decoder at the processor pipeline
- Registers are preserved as 32-bit but only half of them are



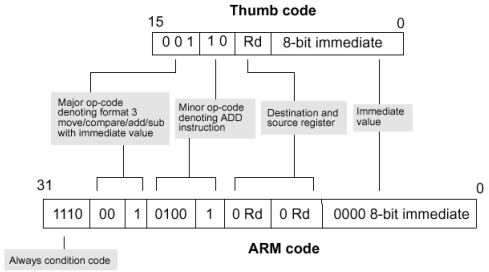


Thumb extension

- Thumb-instruction decoder is placed in pipeline
- Change to Thumb-mode happens by turning the state of multiplexers feeding the instruction decoders and data bus
- A1 selects the 16-bit half word from the 32-bit bus



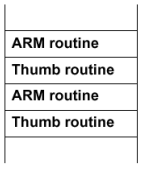
Example: `ADD Rd,#Constant`



- Example of instruction conversion
- Thumb-instruction ADD Rd,#constant is converted to unconditionally executed ARM-instruction ADD Rd,Rn,#constant
- Only the lower register set is in use so the upper register bit is fixed to zero and source and destination are equal. The constant is also 8-bit instead of 12-bit available in ARM-mode

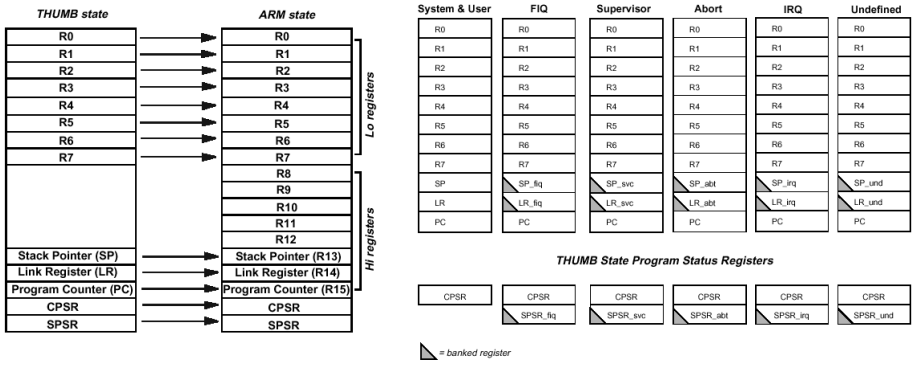
Changing the mode

- Set T-flag in CPSR register and execute BX (Branch eXchange) to the address the thumb code begins at
- Same memory space and contain mixed native ARM-code and Thumb-code
- Execution speed of 32-bit ARM-code decreases significantly if system uses only 16-bit data bus
- If native ARM-code is used, typically it is contained in separate ROM-area as a part of ASIC (ASSP) chip
- Return to Thumb code from native ARM-code can be made by resetting the T-flag and executing BX to desired address



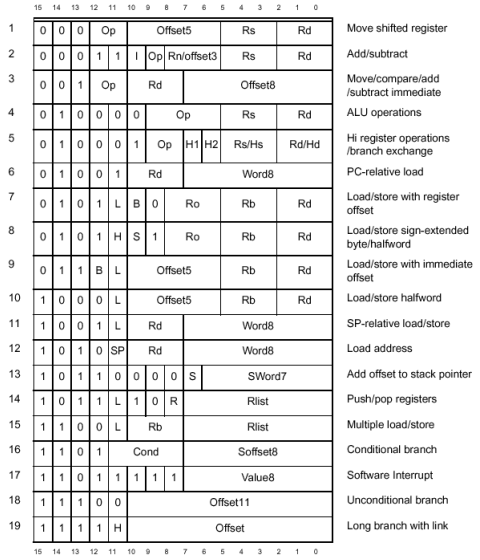
Thumb-state registers

- Only lower part of the register immediately available
- Upper register set (R8-R15) can be used with assembler code
  - Instructions MOV, CMP and ADD are available between register sets



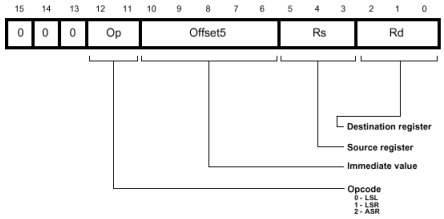
Thumb instruction set

- Instruction word length shrunk to 16-bits
- Instructions follow their own syntax but each instruction has it's native ARM instruction counterpart
- Due to shrinking some functionality is lost
- 19 different Thumb instruction formats

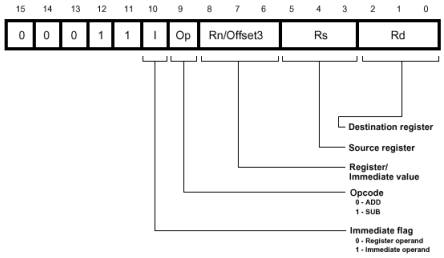


Format 1 and Format 2

- Format 1: Move shifted register
  - LSL, LSR, ASR
  - F.ex. LSL Rd, Rs, #offset shifts Rs left by #offset and stores the result in Rd



- Format 2: Add/subtract
  - ADD, SUB
  - F.ex. ADD Rd, Rs, Rn adds contents of Rn to contents of Rs and places the result in Rd

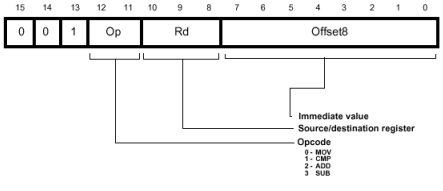


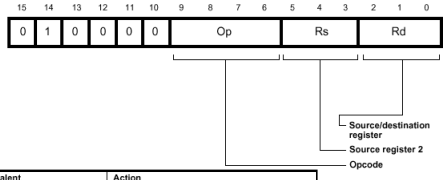
Format 3 and Format 4

- Format 3: Move/compare/ add/subtract *immediate*
  - MOV, CMP, ADD, SUB
  - F.ex. MOV R0, #128
- Format 4: ALU operations
  - 16 different arithmetic / logical operations for registers, see table
  - F.ex.

    MUL R0, R7

    R0 = R7*R0

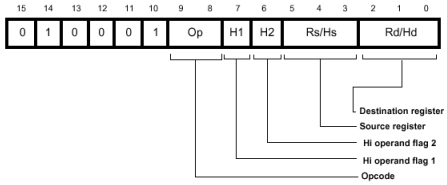




OP	THUMB assembler	ARM equivalent	Action
0000	AND Rd, Rs	ANDS Rd, Rd, Rs	Rd:= Rd AND Rs
0001	EOR Rd, Rs	EORS Rd, Rd, Rs	Rd:= Rd EOR Rs
0010	LSL Rd, Rs	MOVS Rd, Rd, LSL Rs	Rd := Rd << Rs
0011	LSR Rd, Rs	MOVS Rd, Rd, LSR Rs	Rd := Rd >> Rs
0100	ASR Rd, Rs	MOVS Rd, Rd, ASR Rs	Rd := Rd ASR Rs
0101	ADC Rd, Rs	ADCS Rd, Rd, Rs	Rd := Rd + Rs + C-bit
0110	SBC Rd, Rs	SBCS Rd, Rd, Rs	Rd := Rd - Rs - NOT C-bit
0111	ROR Rd, Rs	MOVS Rd, Rd, ROR Rs	Rd := Rd ROR Rs
1000	TST Rd, Rs	TST Rd, Rs	Set condition codes on Rd AND Rs
1001	NEG Rd, Rs	RSBS Rd, Rs, #0	Rd = -Rs
1010	CMP Rd, Rs	CMP Rd, Rs	Set condition codes on Rd - Rs
1011	CMN Rd, Rs	CMN Rd, Rs	Set condition codes on Rd + Rs
1100	ORR Rd, Rs	ORRS Rd, Rd, Rs	Rd := Rd OR Rs
1101	MUL Rd, Rs	MULS Rd, Rs, Rd	Rd := Rs * Rd
1110	BIC Rd, Rs	BICS Rd, Rd, Rs	Rd := Rd AND NOT Rs
1111	MVN Rd, Rs	MVNS Rd, Rs	Rd := NOT Rs

Format 5

- Format 5: Hi register operations / branch exchange

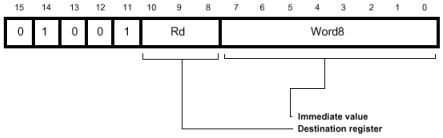


- BX Rs / BX Hs performs a branch with optional mode change. To enter ARM mode, clear bit 0 of Rs before executing the instruction. Thumb mode is entered equivalently by setting the bit.

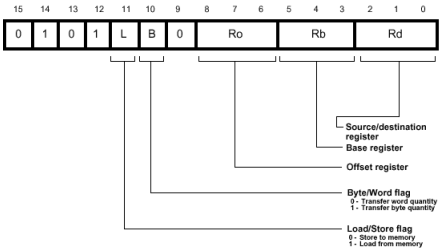
Op	H1	H2	THUMB assembler	ARM equivalent	Action
00	0	1	ADD Rd, Hs	ADD Rd, Rd, Hs	Add a register in the range 8-15 to a register in the range 0-7.
00	1	0	ADD Hd, Rs	ADD Hd, Hd, Rs	Add a register in the range 0-7 to a register in the range 8-15.
00	1	1	ADD Hd, Hs	ADD Hd, Hd, Hs	Add two registers in the range 8-15
01	0	1	CMP Rd, Hs	CMP Rd, Hs	Compare a register in the range 0-7 with a register in the range 8-15. Set the condition code flags on the result.
01	1	0	CMP Hd, Rs	CMP Hd, Rs	Compare a register in the range 8-15 with a register in the range 0-7. Set the condition code flags on the result.
01	1	1	CMP Hd, Hs	CMP Hd, Hs	Compare two registers in the range 8-15. Set the condition code flags on the result.
10	0	1	MOV Rd, Hs	MOV Rd, Hs	Move a value from a register in the range 8-15 to a register in the range 0-7.
10	1	0	MOV Hd, Rs	MOV Hd, Rs	Move a value from a register in the range 0-7 to a register in the range 8-15.
10	1	1	MOV Hd, Hs	MOV Hd, Hs	Move a value between two registers in the range 8-15.
11	0	0	BX Rs	BX Rs	Perform branch (plus optional state change) to address in a register in the range 0-7.
11	0	1	BX Hs	BX Hs	Perform branch (plus optional state change) to address in a register in the range 8-15.

Format 6 and Format 7

- Format 6: PC relative load
  - F.ex. LDR Rd, [PC, #imm] adds unsigned (forward looking) offset (255 words, 1020 bytes) in *imm* to the current value of the PC.

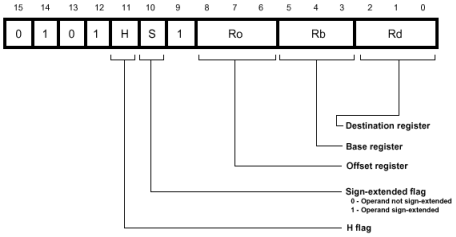


- Format 7: Load/store with register offset
  - LDR, LDRB, STR, STRB
  - F.ex. STR Rd,[Rb, Ro] calculates the target address by adding together Rb and Ro and stores the contents of Rd at the address.



Format 8

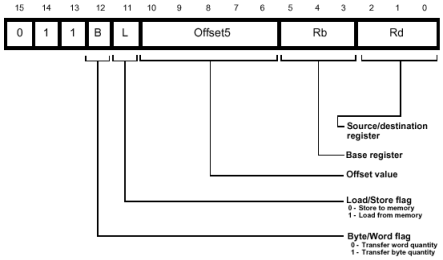
- Format 8: Load / store sign-extended byte / halfword
- LDSB, LDSH, LDRH, STRH



S	H	THUMB assembler	ARM equivalent	Action
0	0	STRH Rd, [Rb, Ro]	STRH Rd, [Rb, Ro]	Store halfword: Add Ro to base address in Rb. Store bits 0-15 of Rd at the resulting address.
0	1	LDRH Rd, [Rb, Ro]	LDRH Rd, [Rb, Ro]	Load halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to 0.
1	0	LDSB Rd, [Rb, Ro]	LDRSB Rd, [Rb, Ro]	Load sign-extended byte: Add Ro to base address in Rb. Load bits 0-7 of Rd from the resulting address, and set bits 8-31 of Rd to bit 7.
1	1	LDSH Rd, [Rb, Ro]	LDRSH Rd, [Rb, Ro]	Load sign-extended halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to bit 15.

Format 9

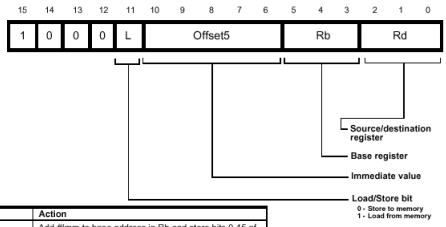
- Format 9: Load / store with immediate offset
- LDR, LDRB, STR, STRB



L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, #Imm]	STR Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the contents of Rd at the address.
1	0	LDR Rd, [Rb, #Imm]	LDR Rd, [Rb, #Imm]	Calculate the source address by adding together the value in Rb and Imm. Load Rd from the address.
0	1	STRB Rd, [Rb, #Imm]	STRB Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the byte value in Rd at the address.
1	1	LDRB Rd, [Rb, #Imm]	LDRB Rd, [Rb, #Imm]	Calculate source address by adding together the value in Rb and Imm. Load the byte value at the address into Rd.

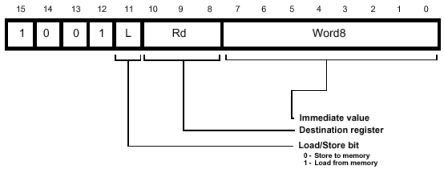
Format 10 and Format 11

- Format 10: Load / store halfword
- LDRH, STRH



L	THUMB assembler	ARM equivalent	Action
0	STRH Rd, [Rb, #Imm]	STRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb and store bits 0-15 of Rd at the resulting address.
1	LDRH Rd, [Rb, #Imm]	LDRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb. Load bits 0-15 from the resulting address into Rd and set bits 16-31 to zero.

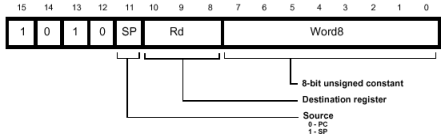
- Format 11: SP-relative load / store
- LDR, STR



L	THUMB assembler	ARM equivalent	Action
0	STR Rd, [SP, #Imm]	STR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Store the contents of Rd at the resulting address.
1	LDR Rd, [SP, #Imm]	LDR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Load the word from the resulting address into Rd.

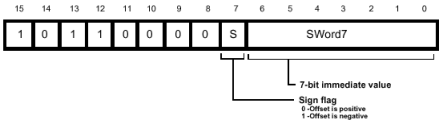
Format 12 and Format 13

- Format 12: Load address



SP	THUMB assembler	ARM equivalent	Action
0	ADD Rd, PC, #Imm	ADD Rd, R15, #Imm	Add #Imm to the current value of the program counter (PC) and load the result into Rd.
1	ADD Rd, SP, #Imm	ADD Rd, R13, #Imm	Add #Imm to the current value of the stack pointer (SP) and load the result into Rd.

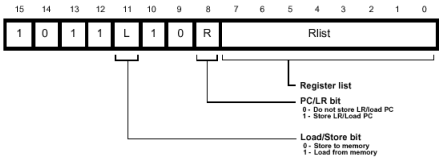
- Format 13: Add offset to Stack Pointer



S	THUMB assembler	ARM equivalent	Action
0	ADD SP, #Imm	ADD R13, R13, #Imm	Add #Imm to the stack pointer (SP).
1	ADD SP, #-Imm	SUB R13, R13, #Imm	Add #-Imm to the stack pointer (SP).

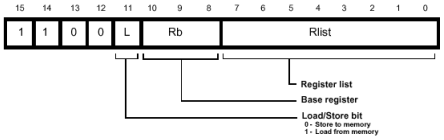
Format 14 and Format 15

- Format 14: Push / pop registers
- PUSH, POP



L	R	THUMB assembler	ARM equivalent	Action
0	0	PUSH { Rlist }	STMDB R13!, { Rlist }	Push the registers specified by Rlist onto the stack. Update the stack pointer.
0	1	PUSH { Rlist, LR }	STMDB R13!, { Rlist, R14 }	Push the Link Register and the registers specified by Rlist (if any) onto the stack. Update the stack pointer.
1	0	POP { Rlist }	LDMIA R13!, { Rlist }	Pop values off the stack into the registers specified by Rlist. Update the stack pointer.
1	1	POP { Rlist, PC }	LDMIA R13!, { Rlist, R15 }	Pop values off the stack and load into the registers specified by Rlist. Pop the PC off the stack. Update the stack pointer.

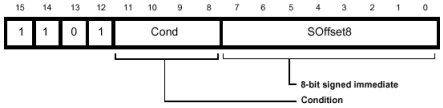
- Format 15: Multiple load / store
- LDMIA, STMIA



L	THUMB assembler	ARM equivalent	Action
0	STMIA Rb!, { Rlist }	STMIA Rb!, { Rlist }	Store the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.
1	LDMIA Rb!, { Rlist }	LDMIA Rb!, { Rlist }	Load the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.

Format 16

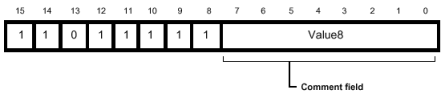
- Format 16: Conditional branch
- BEQ, BNE, BCS, BCC, BMI, BPL, BVS, BHI, BLS, BGE, BLT, BGT, BLE



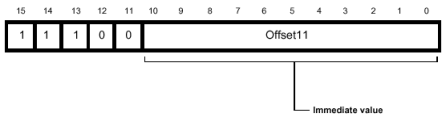
Cond	THUMB assembler	ARM equivalent	Action
0000	BEQ label	BEQ label	Branch if Z set (equal)
0001	BNE label	BNE label	Branch if Z clear (not equal)
0010	BCS label	BCS label	Branch if C set (unsigned higher or same)
0011	BCC label	BCC label	Branch if C clear (unsigned lower)
0100	BMI label	BMI label	Branch if N set (negative)
0101	BPL label	BPL label	Branch if N clear (positive or zero)
0110	BVS label	BVS label	Branch if V set (overflow)
0111	BVC label	BVC label	Branch if V clear (no overflow)
1000	BHI label	BHI label	Branch if C set and Z clear (unsigned higher)
1001	BLS label	BLS label	Branch if C clear or Z set (unsigned lower or same)
1010	BGE label	BGE label	Branch if N set and V set, or N clear and V clear (greater or equal)
1011	BLT label	BLT label	Branch if N set and V clear, or N clear and V set (less than)
1100	BGT label	BGT label	Branch if Z clear, and either N set and V set or N clear and V clear (greater than)
1101	BLE label	BLE label	Branch if Z set, or N set and V clear, or N clear and V set (less than or equal)

Format 17 and Format 18

- Format 17: Software interrupt
- SWI *value8*
- Used to enter interrupt routine (svc mode) pointed by contents of address 0x8. Interrupt service is executed in ARM-state.

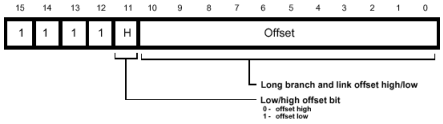


- Format 18: Unconditional branch
- B *label*, ARM equivalent BAL



Format 19

- Format 19: long branch with link
- BL *label*
- 32-bit instructions in two half words: Instruction 1 (H=0) contains the upper 11 bits of the target address. Instruction 2 (H=1) contains the lower 11 bits of the target address.

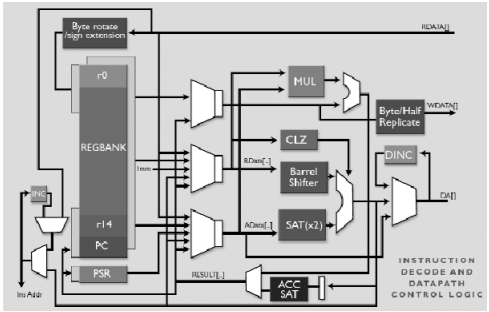


ARM9(TDMI)

- ARM7 microarchitecture is getting old and will be replaced with ARM9
- ARM9 realizes the same (v4T) instruction set that ARM7 and is thus binary compatible
- Pipeline length is 5 stages instead of ARM7 3 stages. This allows for faster clocking.
- Available with TDMI extensions
- ARM92x: ARM9TDMI and caches as a macrocell
- Caches are separate for instructions and data (Harvard-architecture)

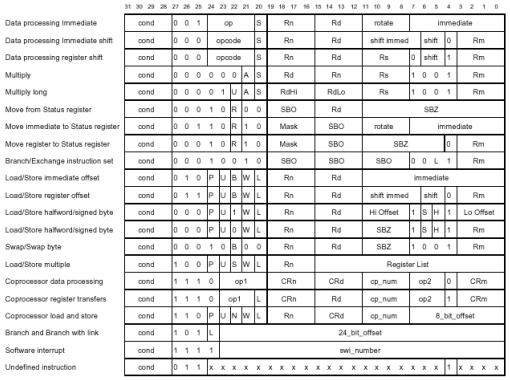
ARM9

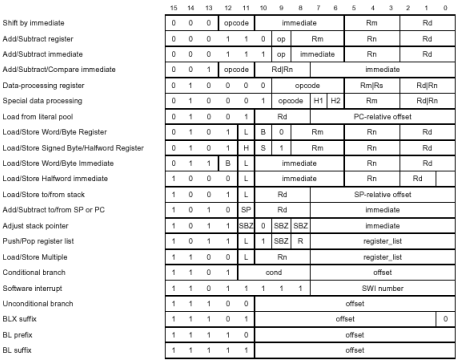
- ARM9 claims 143 MIPS@130 MHz -> more than one instruction per clock cycle -> not explained with pipeline modification, must have increased parallelism?



ARM10(TDMI)

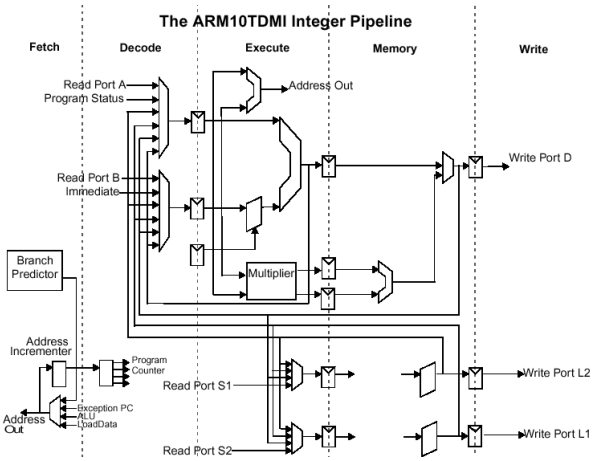
- ARM10TDMI processor core:
  - Realizes the ARM instruction set with binary compatibility including the Thumb extension
  - Instruction set expanded to version 5 (v5TE), 32x16 MAC-multiplier
  - 6-stage pipeline for fixed point instructions





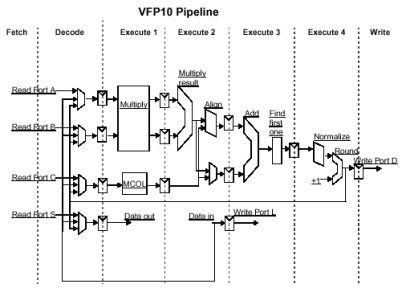
ARM10

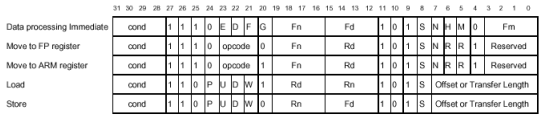
- Improved instruction execution:
  - Added parallelism, branch prediction, 64-place TLB (Translation Look aside Buffer), parallel store/load unit, caches
  - Claims 400 MIPS @ 333 MHz (ARM1020TE macrocell, 32+32kB caches, 10 mm2 / 0.15u 5 metal layer process, <0.85 mW / MHz)



ARM extensions: VFP10

- VFP10 i.e. Vector Floating Point Processor
  - Floating point extension to ARM10, IEEE-754:n compliant
  - 7-stage ALU-pipeline, 5-stage load/store-pipeline
  - single and double precision operations, 32 SP registers on top of 16 DP-registers





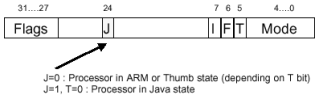
VFP instruction set

Mnemonic	Operation	Mnemonic	Operation
FADD	Add	FCPY	Copy (Move)
FSUB	Subtract		
FMUL	Multiply	FNMUL	Negated multiply
FMAC	Multiply-Accumulate	FNMAC	Negated Multiply-Accumulate
FMSC	Multiply-Subtract	FNMSC	Negated Multiply-Subtract
FDIV	Divide	FSQRT	Square Root
FABS	Absolute Value	FNEG	Negate
FCMP	Compare Register with Register	FCMPZ	Compare with Zero
FCVTDS	Convert Double to Single	FCVTSD	Convert Single to Double
FITOF	Convert Integer to Float	FFTOI	Convert Float to Integer
FLDR	Load Single Value	FSTR	Store Single Value
FLDM	Load Multiple Values (Vector)	FSTM	Store Multiple Values (Vector)

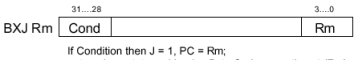
ARM extensions: Jazelle

- Jazelle = Java-bytecode executing extension, in practice adds third instruction set to an ARM-processor core
- New Java operating mode:

1) Addition of 'J' bit to CPSR:



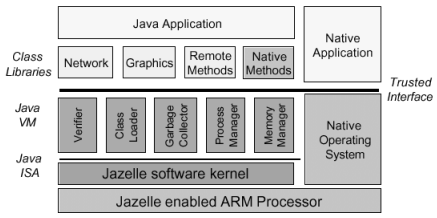
2) New ARM instruction:



If Condition then J = 1, PC = Rm;
enters Java state and begins Byte Code execution at (Rm)

- 140 Java-instructions are executed directly in hardware, rest 94 by emulating with multiple ARM-instructions

- Example of software architecture:



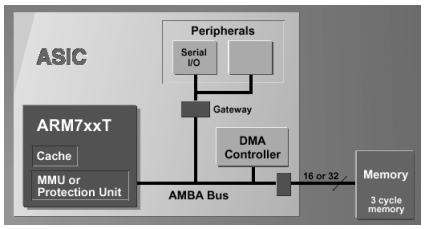
Use of ARM Registers in Jazelle State:

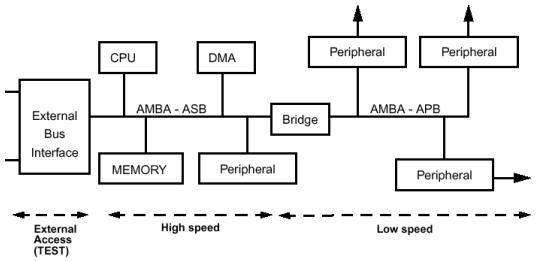
R0-R3	Used to cache Java expression stack
R4	Local variable 0 ('this' pointer)
R5	Pointer to table of SW handlers
R6	Java stack pointer
R7	Java variables pointer
R8	Java constant pool pointer
R9-R11	Reserved for JVM (not used by h/w)
R12, R14	Scratch usage / Java return address
R13	Machine stack pointer
R15	Java PC

- Context switching?

ARM extensions: AMBA

- AMBA-bus:
- ASB i.e. AMBA System Bus
- APB i.e. AMBA Peripheral Bus
- AHB i.e. AMBA High bandwidth Bus



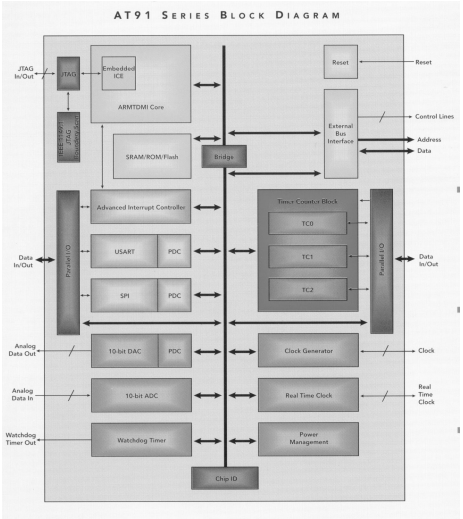


ARM as a standard component

- Even tough ARM is mostly used as a processor core in SoC and other ASICs have some manufacturers brought ARM-based standard products to market
- Examples of manufacturers: Atmel, Cirrus Logic, Hyundai, Intel, Oki, Samsung, Sharp …
- Most of the products are based on 7TDMI-core, some to 720T- and 920T-cores
- ARM + FPGA: Altera and Triscend
- In addition, there are a number of ASSP (Application Specific Standard Product) -chips available for example to communication applications (Philips VWS22100 = ARM7 based GSM baseband chip).

Atmel ARM

- AT91-series:
  - ARM7TDMI-core
  - External bus controller
  - A load of peripherals
  - Variable amount of SRAM on die (up to 2 megabits)



Altera ARM + FPGA

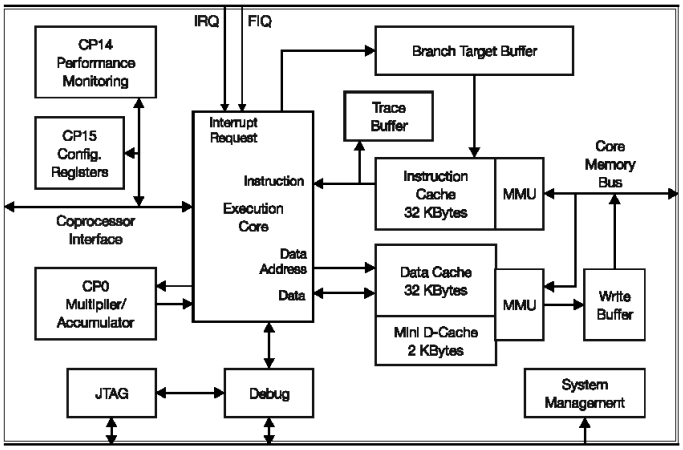
- ARM922T macrocell and programmable logic on same chip
  - System-on-a-programmable-chip

Table 1. Current ARM-Based Embedded Processor Device Features (1)

Feature	EPXA1	EPXA4	EPXA10
Maximum system gates	263,000	1,052,000	1,772,000
Typical gates 100,000	400,000	1,000,000	
Logic elements (LEs)	4,160	16,640	38,400
Embedded system blocks (ESBs)	26	104	160
Maximum RAM bits	53,248	212,992	327,680
Maximum macrocells	416	1,664	2,560
Maximum user I/Os	178	360	521
Single-port SRAM	32 Kbytes	128 Kbytes	256 Kbytes
Dual-port SRAM 16 Kbytes	64 Kbytes	128 Kbytes	

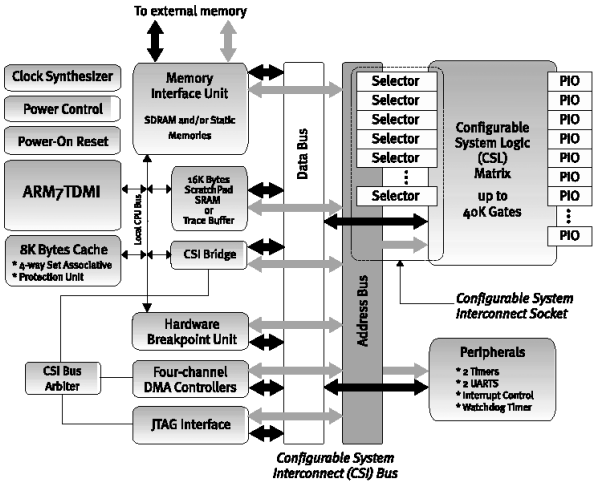
Intel ARM derivatives

- StrongARM
  - DEC developed ARM variant
  - Being phased out with XScale
- XScale
  - ARM v5TE instruction set
  - Intel developed microarchitecture
  - Coprocessor instructions used for extensions



Triscend ARM + FPGA

- Triscend A7:



Block Diagram of the Triscend A7 Configurable System-on-Chip

What does it look like on silicon?

- ARM7TDMI
- 5kB SRAM
- 130k ports of logic
- USB-port