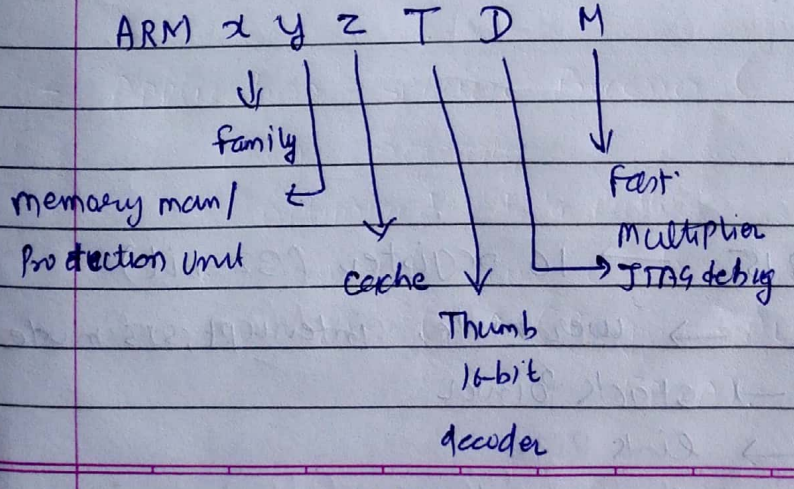
ARM PROCESSOR

- invention of RISC → ARPA's VLSI project
  ↳ (advanced research Proj Agax
- ARM → acorn RISC machine
- low power and Peformance is good.

- RISC :
  - fixed and same length of instructions.
  - Pipelining
  - reduces the instructions
  - large general purpose register
  - load / store

ARM : ARM Processor has been specially designed to be small to reduce power consumption and extented battery operations.
- it's provides higher coding density
- ARM core is not a pure RISC, because of its primary application — the embedded sys.
- Price sensitive

ARM nomenclature :



CPSR → SWI → software interrupt

Cortex – A , M , R

↓ → Real time

Application

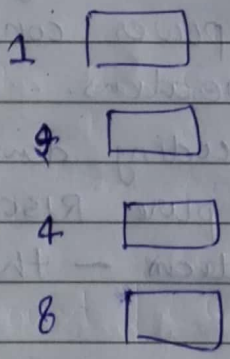
Microcontroller

- Cache tightly coupled } → SRAM Memory

  ↳ For particular Application → memory addr are fixed (as 0 to 4 if required)

Barrel ~~Barattel~~ shifter

- time reduce



diff ways to attach the data → addr mode

Registers

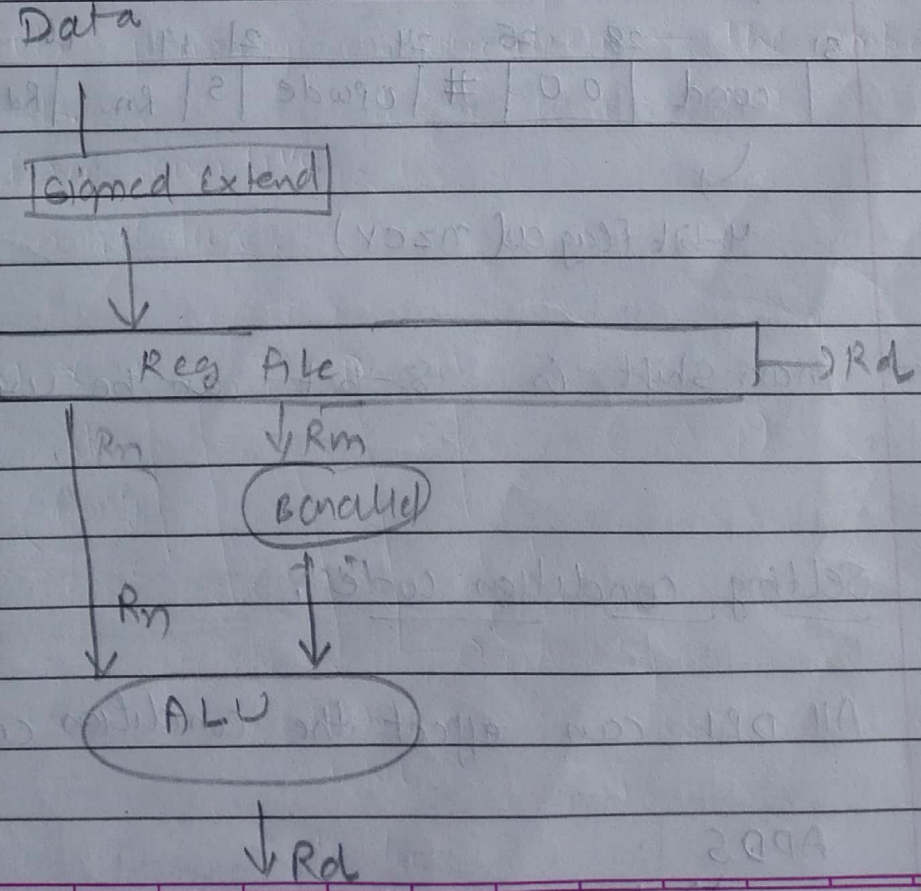
R0 – R15 → 16 register (32-bit)

general Purpose
- R0-R12 → user mode, interrupt, SVC mode
- R13 → stack Pointer
- R14 → link Reg.
- R15 → Prog. counter

- CPSR
(current prog. status ~~Reegram~~ Register)
- Thumb mode
- Additional register (SPSR) → (save special prog status Reg.)
- SPSR and cond in 32-bit register both are updated by using 's'

adds

ARM Data flow

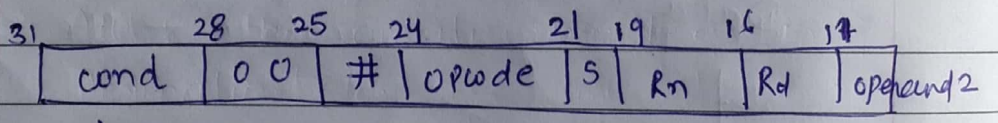


Barrel Berattel shifter → shift Right / left all types of shifting operation is done by barrel shifter.

Types of Instructions:

(A) Data Processing Instr
- Register movement
- ALU
- Bit wise
- Comparison

* n z c v
↓ ↓ ↓ ↳ overflow } flag
neg. zero carry



↳ 4-bit flag (n z c v)

- max. shift is 8-bit can be used.

Setting condition code:

- All DPI can affect the condition codes

ADDS .
↳ to upload the conditions on flag in

Assembly prog. / Language.

- CPSR

F → fast interrupt
0 - enable
1 - disable

I → interrupt
0 ← enable
1 ← disable

T → set 0 bit for 16-bit taken
1 → 32-bit

⇒ 8 - Arm

Advanced Microcontroller Bus Architecture

AHB ← (advanced high Bus)
- arch.

Interrupt controller

APB (advanced Peripheral Bus)

Mem. controller

AHB & APB interface → using AHB APB Bridge

ARM7 TDMI :

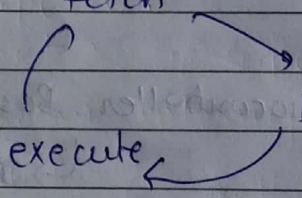
- Version 4
- Thumb
- Debug (on chip)

} TDMI

- enhance multiply (DSP etc)
- Embedded ICE hardware
- Von-neumann arch.
  - 32 bit data bus
  - words - 4 byte
  - Half word - 2 byte
  - Data size - byte, word, haft etc

RISC
- load & store arch. in memory
- R, I, J fixed length instruction format
- Opcode (destination, source 1, source 2)
- inline barrel shifter
- Variable cycle instruction.

Instruction cycle :



ALU & other operation based on instr, destination etc.

- ARM supports both little endian and big endian memory format.
  0-15 mem. registers

Modes: in CPSR

① User Mode
- R13 — Stack Pointer
- R14 — Link register (like return add store in RISC)

for jump or branch instr
R15 → Prog. counter
CPSR → current prog. status Register

(2) Interrupt Request mode (IRQ)
R13, R14 & SPSR (save prog. status reg.)

(3) FIQ (fast interrupt Request) mode
R8 -14, SPSR (0-12) data, allows faster data tx than normal.
(for internal bus channel)

(4) Undefined mode (not in predefined formal)
R13 -14, SPSR

(5) Abort mode
R13 -14, SPSR

- instruction fetch Abort — Data not proper
fetch data — add' of
bcoz addres — data memory
not current

(6) (SUC) System mode - R13, 14 & SPSR
abnormal behaviours or first configuration
R0 - R7 → general Purpose

(7) Supervisor mode

* CPSR:

all are 8 bit

flags	status	extension	control
NZCV			(IFT) Mode

IR → interrupt ~~mode~~ fast

I → 0 normal interrupt mode is enable
1 not enabled

F → 1 → disable
0 → enable

T → thumb mode (1-bit instruction)
0 → enable
1 → disable

mode → processor mode

SPSR → same arch as CPSR

Exceptions:

priorities ↓

1. Reset (supervisor mode)
2. Software interrupt-(" ")
3. IRQ } ~~interrupt~~ mode
4. FIQ
5. Data abort (abort mode)
6. Undefined instr
7. Prefetch abort (abort mode

→ fixed addr stmd in IVT.

Arm Condition codes :

OPcode [31:28]		
0000	EQ	equal
0001	NE	not equal

Page No.:
Date: 10 | Augo

CS — carry set
CC — Carry clear

- Register Movement operations:
  - MOV $r0, $r2 // $r0 = $r2
  - MOVN $r0, $r2 — $r0 = !$r2
    ↳ not equal
  - SUB $r0, $r1, $r2 — $r0 = $r1 - $r2
  - RSB (reverse subtraction)

- Bit wise
  - EOR → ex-or
  - BIC → $r0 = $r1 and (not) $r2

- Comparison
  - CMP $r1, $r2 (compare) — flags set as a rst of $r1 - $r2
    - $r1 = $r2 ; zero flag
    - $r1 > $r2 ;
    - $r1 < $r2 ;
  - CMN $r1, $r2 — $r1 + $r2 → flag set
  - TST $r1, $r2 — flag set as a result of $r1 and $r2 (bit wise)
  - TEQ $r1, $r2 — $r1 XOR $r2

- immediate operands
  - Add $r3, $r3, #1 ; $r3 = $r3 + 1
    - #&FF
      ↳ max. (4 bit 24 = 16)

- shift register Operands

ADD r3, r2, r1, LSL #3

↳ Logical shift left

; r3 = r2 + (r1 * 8)

By shifting left by 3

- Second operand must goes into the Barrel shifter

LSL - logical shift left RRX

LSR - " " Right (rotate

ASL - Airthmetic " left

ASR - " " Right

ROR → Rotate Right (used for swapping)

- LSL & ASL are same bcoz we are 0 extented into the LSB
- LSR → 0 extented into the MSB

ASR → MSB is same (bcoz of sign bit)

Multiplication:

MUL r4, r3, r2

- immediate second operand not supported
- result register is not same as the first source

MULA r4, r3, r2, r1 multiplying accumulation

r4 = (r3 * r2) + r4

Ex:- r0 = r1 + (r1 * 4)

add r0, r1, r1, LSL #2

Ex:- r2 = r3 * 119

= r3 * 17 * 7

= r3 * (16+1) * (8-1)

Add r2, r3, r3, LSL #4

Add r2, r2, r2, LSL #3

→ Second operand is immediate value, that will have to convert as example.

- MULL which gives RdHi, RdLo = Rm * Rs
- MUAL

(when Rm, Rs } 32bit

22-Aug.

Branch Instructions:

Branch ⟶ B

Branch with link ⟶ BL

Conditional Branch: (conditional jump in 8086)

B same as jump ⟶ Unconditional

BEQ ⟶ equal (Check zero flag)

BLO ⟶ Unsigned comp. gave lower

BLS ⟶

load store instruction:

three types of load - store inst^{r},

i) single register transfer
ii) Multiple " "
iii) swap

supported data type → 8 bit or 16 or 32

- LDR → load word into a register
- STR → store word from " "

addressing mode:

(i) – pre indexing

LDR r0, [r1, #4]

r1 is not change

r0 = mem[r1+4]

r1 = r1

(ii) – Post indexing

LDR r0, [r1], #4

r0 = mem[r1]

r1 = r1 + 4

(iii) – Auto indexing (Pre index write back)

LDR r0, [r1 #4]!

r0 = mem[r1+4]

r1 = r1+4

By using Post indexing complexity is less and code length is less. So we are using this in ARM. But we can't used this in RISC or 8086.

- Multiple register transfer

- LDM IA r1, {r0, r2, r5}
  - load multiple
  - IA → After increment

r0 = mem[r1]
r2 = mem[r1+4]
r5 = mem[r1+8]

- LDM DA r1, {r0-r11}
  - load multiple
  - DA → after Decrement

r0 = mem[r1]
r1 = mem[r1+4]
r5 = mem[r1-8]

register which are defined ~~only~~ for User mode they are only used.

- loading 32-bit constant
  - directly its not possible
  - LDR Rd → is the Pseudo for ARM
  - literal Pool → constant data area in the embedded sys

Software interrupt instruction (SWI).

- SWI → SVC mode
  - lr_svc = addr of instr following the swi
  - SPSR_SVC = CPSR
  - PC = vectors + 0x8
  - CPSR mode = SVC
  - CPSR I = 1 (mask IRQ interrupts)

- MRS ~~register~~ instruction } To store the CPSR and SPSR values in another register
(Copp. to MRS) MSR "

- CDP → coprocessor data processing
CDP, MRS, MSR
LDC STC

coprocessor instructions

ARM V5E extention

CLZ → Count leading zero
CLZ r0, r1
r1 = 0b 0000000000000000000000001
fill MSB → count no. of zero included MSB
r0 = ___ (no of zeros)

ADDS → saturated arithmetic
ADDS r0, r1, r2
Negative flag is active → overflow flag is not active → Q is active
Q ADD → saturation is active
QADD Rd = Rn + Rd
QDADD Rd = Rn + (Rd * 2)

SMLATB → Signed multiply with accumulate to byte

SMLATB r0, r1, r2, r3

r1 = 0b 80000001

r2 = 0b 00000001

r3 = 0b 00000004

r1 → upper 4, r2 → lower

$$r0 = \begin{cases} 8 \times 1 \rightarrow 8 \\ 8 + 4 \rightarrow 12 \end{cases}$$

r0 = 0b 00000012

---

if {(a==b) && (c==d)} e++

CMP

CMP r0, #5

BEQ bypass

⇒ ADDNE r0, r1, r2

SUBNE r0, r1, r2

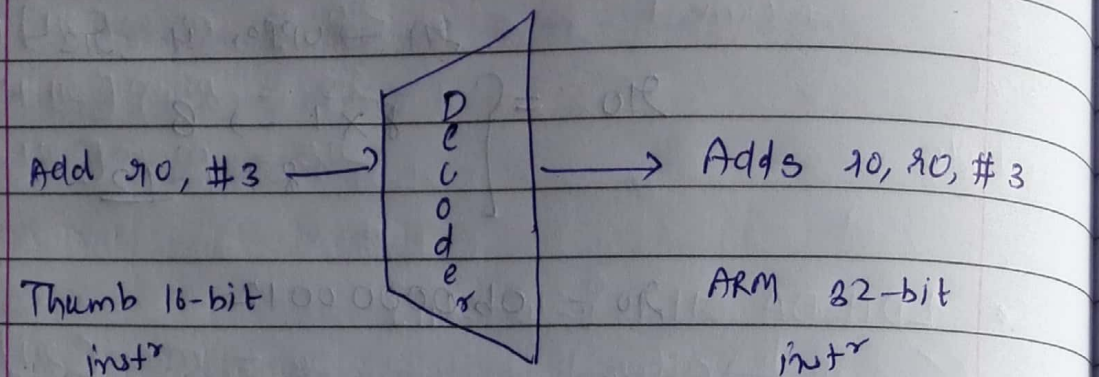
from side

By using ARM, Branch inst^{n} can be replaced → so code density ↑s → dependency is also removed so we can say that performance ↑es.

Page No.:
Date: 24/Aug

Thumb instr set:

- High code density



- code density uses
- we required 2 byte for per instr in memory
- we required 4 byte location in memory for instr.

- MSR and MRS are not valid in thumb instructions.
- 8 bit −256 to +254 → after condition statement is satisfied.
- Offset limit of immediate value is 5 bit in load / store instr.
- LDMIA → only allowed in thumb
- PUSH, POP → are ~~allowed~~ possible in thumb
- r0 to r7 → acess in thumb
- Cmp and move can only used the r8 to r13.
- thumb mode dont support the coprocessor

Page No.:
Date: 5 | Sept

Real Time Operating System

Embedded

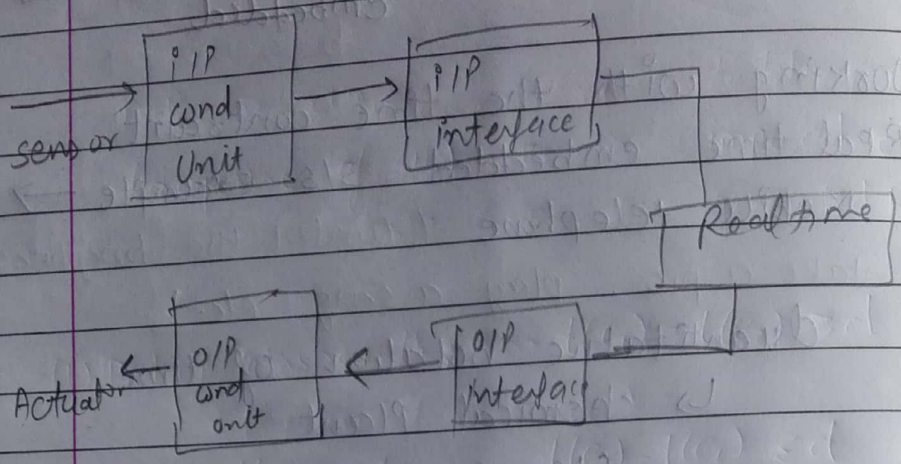
- Working with the time contraint
- Real time embedded s/s example →
  Mobile telephone (control the brightness, take a pic, play a song etc.)
- Quantitative notation of time
  - ↳ chemical Plant
    (temp^r is very crucial)
  - ↳ nuclear Plant

- Qualitative notation of time
  - ↳ Rain (if rain is there)
  - ↳ library

- Real time means times as prescribed by external sources. It's quantitative notation of time

- Application
  - Chemical Plant control
  - Automatic car Assembly Plant
  - Supervisory control and data acquisition
  - laser Printer – peripherals equipment
    (non critical s/s
  - Multy fuel injection s/s – Automobiles
    Point

Basic Model of RTES:



Charc of RTES:

(I) – Time Conbaints → "deadline" (task schedulin
(II) – New correctness criteria
- Not only logical correctness but also in time

(III) – embedded
(IV) – Safety - criticality
safe sis does not cause any damage even when its fail.
- Safety and Reliability are together

(V) – Concurrency
(VI) – Distributed and feedback Structure
(VII) – Task criticality
- measure of the cost of failure of a work.

(VIII) – Custom hardware
(IX) – reactive
(X) – Stability
(XI) – Exception handling

How to achieve Reliability :

- Error avoidence
- error detection and removal
- fault tolerance

Built in self test (BIST)

Triple modular redundancy (TMR)

Software fault tolerance

N-version programming
- satistical correlation of failure

Recovery Blocks
- try Blocks are used

check point and rollback recovery

Types of Real time tasks

(I) HIW RTT
(II) Software RTT (it has time bound)
(III) Non-real time task (task is not associated with time)
(IV) firm real time task

Timing Constraints :

- Event in real time
  - Stimulus (environment)
  - Response (response event at an enviroment)

(Chp-10)

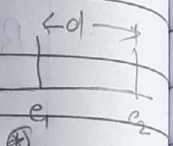
· Rajib mall → Books

www.nawneet.com
Page No.:
Date: 6 | Sepo

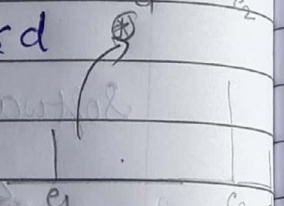
Classification timing constraints:

- Performance (response of S/S)
- Behavioral imposed on stimuli generated by environment

(i) Delay constraints: $t(e_1) - t(e_2) \geq d$



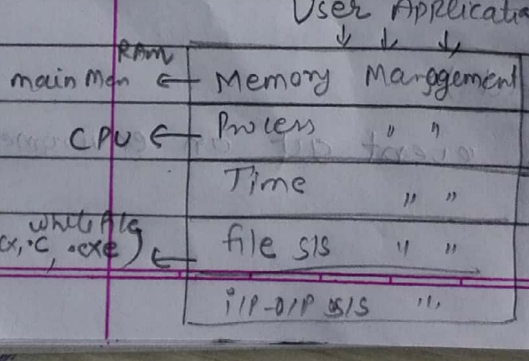
(ii) Deadline : $t(e_2) - t(e_1) \leq d$



(iii) Duration constraints:
Min and max
average time delay is acceptable.

Why OS required for embedded S/S ?
– Draw back of Super loop (infinite loop)

embedded O.S
- Assign the priority to task
- Dynamically change the priority of task
- Schedule the execution of task based on priorities
- Switch the execution



operating S/S architecture

Page No.:
Date: 12 | Sep.

Kernal:

Kernal →

process management:

- managing Process
- setting up memory space for process
  - worst (required 10 mb, we given 100 mb)
  - Best
- scheduling and managing
- primary management unit
  - Volatile memory (RAM)
  - memory management Unit (MMU)

Kernal

Monolithic Kernel Micro Kernel