ARO - Architecture des Ordinateurs

Leonard Cseres - Avril 2024

Hex	Bytes (B)	Bits (b)
0x400	$1\text{KB} = 2^{10}\text{B} = 1024\text{B}$	8Kb
0x800	$1\text{MB} = 2^{20}\text{B} = 1024\text{KB}$	8Mb
0x1000	$1\text{GB} = 2^{30}\text{B} = 1024\text{MB}$	8Gb
0x2000	$1\text{TB} = 2^{40}\text{B} = 1024\text{GB}$	8Tb

Conversion hexadécimal → binaire

0x0	0000	0x6	0110	0xC	1100
0x1	0001	0x7	0111	0xD	1101
0x2	0010	0x8	1000	0xE	1110
0x3	0011	0x9	1001	0xF	1111
0x4	0100	0xA	1010		
0x5	0101	0xB	1011		

Non-Signé

`ADD` $> 2^n - 1$ → C (*carry*)

`SUB` < 0 → B (*borrow*)

Signé

$< -2^{n-1}$ ou $> 2^{n-1} - 1$ → V (*overflow*)

CPU (Contrôle/traite/execute)

Mémoire ⟺ Unité de **contrôle** | Unité de **traitement** (ALU/Registres)

Mémoire

volatile (RAM/SRAM) | non-volatile (flash memory)

Von Neumann Architecture avec un seul bus pour les données et les instructions

Harvard Architecture avec deux bus séparés pour les données et les instructions

Adressage $A_{fin} = A_{debut} + T - 1$ où T est la taille de la zone d'adressage

Endianness *Big Endian* (octet de poids fort en premier), *Little Endian* (octet de poids faible en premier)

Alignement (Mem. 64-bits)

	0	1	2	3	4	5	6	7
00000000	00	00	00	00	00	00	00	00
00000008	00	00	00	00	00	00	00	00
00000010	00	00	00	00	00	00	00	00
...	...	...	...	...	...	...	...	...

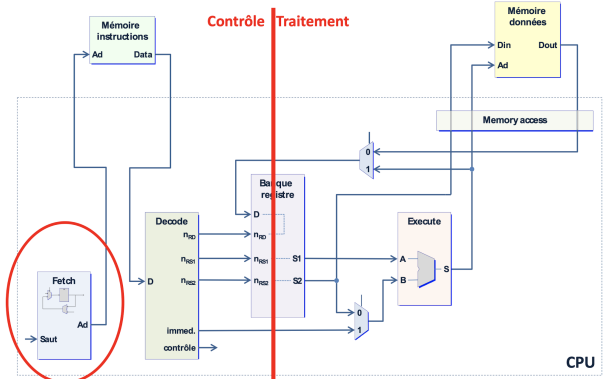
64 bits word | 32 bits word | 16 bits word

Types d'Instructions

- Instruction de **calcul/traitement**
- Instruction de **transfert de données**
- Instruction de **contrôle** (branchement, saut, interruption)

Registres accès rapide (plus que la mémoire), stockage temporaire, passage de paramètres, sauvegarde de contexte

Registres	Fonction
`SP`	Pointeur de pile
`LR`	Registre de lien (appel de fct, interruption)
`PC`	Compteur de programme
`CPSR`	Registre d'état



Fetch Récupération de l'instruction à exécuter

PC Correspond au nombre de bytes de l'instruction, *indépendamment du bus d'adresses*

Bits d'instruction	8 bits	16 bits	32 bits
Incrément	1	2	4

Sauts $A_{fut} = A_{PC} + \texttt{extension\_16bits}(offset_{11} \cdot 2) + 4$ avec $offset_{11}$ signé et `extension_16bits` garde le signe

Vecteur d'interruption $A_{vec} = A_{base} + (C \cdot N)$ avec C le numéro de l'interruption et N le nbr. de bytes du bus d'instructions

ISR	Adresse Vecteur
0x00000	A_{base}
...	$A_{base} + (1 \cdot N)$

Decode Décodage de l'instruction pour déterminer l'opération à effectuer

CSPR

- `N` (Negative) : résultat négatif
- `Z` (Zero) : résultat nul
- `C` (Carry) : retenue
- `V` (Overflow) : dépassement
- `IF` (Interrupt Flag) : autorisation d'interruption
- `T` (Thumb) : jeu d'instructions
- `M` (Mode) : mode d'exécution

Utilisation de la pile `PUSH` et `POP`

- Stockage des adresses de retour
- Sauvegarde des registres
- Stockage des variables locales

ARM Thumb

Mnemonic	Instruction
`ADC`	Add with carry
`B`	Unconditional branch
`Bxx`	Conditional branch
`BL`	Branch with link
`BX`	Branch and exchange
`CMN`	Compare negative
`LDR`	Load word
`LDRB`	Load byte
`LDRH`	Load halfword
`ASR`	Arithmetic shift right
`LSL`	Logical shift left
`LSR`	Logical shift right
`EOR`	Exclusive OR

Pipeline Division du processus d'exécution en plusieurs étapes

- **Fetch** : récupération de l'instruction
- **Decode** : décodage de l'instruction
- **Execute** : exécution de l'instruction
- **Memory** : accès à la mémoire
- **Write-back** : écriture du résultat

Calcul de performance $IPC = \frac{\text{nbr instr base}}{\text{nbr instr réelles}}$, $T_{ex} = \frac{1}{f}$

Latence Temps nécessaire pour exécuter une instruction soit $T_{lat} = m \cdot T_{ex}$ et (1) *Sans pipeline:* $T_{tot} = n \cdot m \cdot T_{lat}$, (2) *Avec pipline:* $T_{tot} = (n + m - 1) \cdot T_{lat}$ avec n le nombre d'instructions et m le nombre de stages

Débit Nombre d'instructions exécutées par unité de temps soit $D = \frac{1}{T_{ex}}$ où $T_{ex} = \frac{T_{lat}}{m}$

Accélération Nombre de fois plus vite qu'en séquentiel soit $A = \frac{T_{seq}}{T_{pip}}$

Aléas

Structurel Ressources partagées

Load	F	D	E	M	W			
I2		F	D	E	M	W		
I3			F	D	E	M	W	
I4				F	D	E	M	W

Données Dépendance entre les instructions

`ADD r1, r2, r3`	F	D	E	M	W		
`SUB r4, r2, r3`		F	D	E	M	W	

`ADD r1, r2, r3`	F	D	E	M	W				
`SUB r4, r2, r3`		-	-	-	F	D	E	M	W

- **RAW** : Read After Write
- **WAR** : Write After Read
- **WAW** : Write After Write

Contrôle Sauts conditionnels (on sait si on fait le saut à partir de `MEM` soit 3 cycles)

Forwarding Transmission directe du résultat d'une instruction à une autre sans passer par la mémoire

`LDRH r1, [r2]`	F	D	E	M	W	
`SUB r4, r1, r3`		F	D	E	M	W

`LDRH r1, [r2]`	F	D	E	M	W				
`SUB r4, r1, r3`		F	D	D	E	M	W		
`AND r5, r1, r6`			F	F	D	E	M	W	
`OR r7, r1, r8`					F	D	E	M	W

Démarche

1. Remplir pipeline
2. Remplir registre E (cycle après execute)
3. Remplir registre M quand il est modifié (cycle après memory access)
4. Remplir registres F1 et F2 en diagonal depuis le registre E
5. Combler registre M avec ce qu'il y avait dans E (au cycle précédent)
6. Identifier les dépendances
7. Remplir états des registres de forwarding

- `LDR` : Mettre des parenthèses
- Les valeurs des registres se répètent en cas d'arrêt