Contents

Carte de Référence ISA A32

Registres

Registre	Alias	Usage
R0-R3	-	Arguments / Retour (caller-saved)
R4-R11	-	Variables (callee-saved)
R12	IP	Scratch
R13	SP	Stack Pointer
R14	LR	Link Register

Registre	Alias	Usage
R15	PC	Program Counter

Drapeaux (NZCV)

Flag	Nom	Signification
N	Negative	Bit 31 du résultat = 1
Z	Zero	Résultat = 0
C	Carry	Retenue sortante
V	oVerflow	Débordement signé

Conditions

Cond	Signification	Flags testés
EQ	Equal	Z=1
NE	Not Equal	Z=0
CS/HS	Carry Set / Unsigned	C=1
CC/LO	Carry Clear / Unsigned <	C=0
MI	Minus (negative)	N=1
PL	Plus (positive or zero)	N=0
VS	Overflow Set	V=1
VC	Overflow Clear	V=0
HI	Unsigned >	C=1 et Z=0
LS	Unsigned	C=0 ou Z=1
GE	Signed	N=V
LT	Signed <	N V
GT	Signed >	Z=0 et N=V
LE	Signed	Z=1 ou N V
AL	Always	(défaut)

Instructions ALU

Instruction	Syntaxe	Description
ADD	ADD Rd, Rn, Rm/imm	Rd = Rn + Op2
SUB	SUB Rd, Rn, Rm/imm	Rd = Rn - Op2
RSB	RSB Rd, Rn, Rm/imm	Rd = Op2 - Rn
AND	AND Rd, Rn, Rm/imm	Rd = Rn & Op2
ORR	ORR Rd, Rn, Rm/imm	Rd = Rn | Op2
EOR	EOR Rd, Rn, Rm/imm	Rd = Rn ^ Op2
BIC	BIC Rd, Rn, Rm/imm	Rd = Rn & ~Op2
MOV	MOV Rd, Rm/imm	Rd = Op2
MVN	MVN Rd, Rm/imm	Rd = ~Op2
CMP	CMP Rn, Rm/imm	Flags = Rn - Op2
CMN	CMN Rn, Rm/imm	Flags = Rn + Op2
TST	TST Rn, Rm/imm	Flags = Rn & Op2
TEQ	TEQ Rn, Rm/imm	Flags = Rn ^ Op2
MUL	MUL Rd, Rn, Rm	Rd = Rn * Rm

Ajouter `.S` pour mettre à jour les flags (ex: ADDS)

Instructions Load/Store

Instruction	Syntaxe	Description
LDR	LDR Rd, [Rn, #off]	Rd = Mem[Rn + off] (32 bits)
STR	STR Rd, [Rn, #off]	Mem[Rn + off] = Rd (32 bits)
LDRB	LDRB Rd, [Rn, #off]	Rd = Mem[Rn + off] (8 bits)
STRB	STRB Rd, [Rn, #off]	Mem[Rn + off] = Rd (8 bits)
LDR	LDR Rd, =value	Rd = value (via literal pool)
PUSH	PUSH {regs}	Empiler registres
POP	POP {regs}	Dépiler registres

Instructions de Branchement

Instruction	Syntaxe	Description
B	B label	PC = label
BL	BL label	LR = PC+4; PC = label
BX	BX Rm	PC = Rm

Instructions Système

Instruction	Syntaxe	Description
SVC	SVC #n	Supervisor Call (n=0 pour exit)
NOP	NOP	No Operation

Directives Assembleur

Directive	Description
.text	Section code
.data	Section données initialisées
.bss	Section données non-initialisées
.word val	Mot 32 bits
.byte val	Octet
.asciz “s”	Chaîne avec \0
.align n	Aligner sur 2^n
.ltorg	Placer literal pool ici
.global sym	Exporter symbole

Memory Map

Adresse	Usage
0x00000000	Début RAM
0x00100000	Fin RAM (1 MiB)
0x00400000	Framebuffer
0xFFFF0000	MMIO Output
0xFFFF0004	MMIO Input
0xFFFF0010	MMIO Exit

Encodage (32 bits)

```
[31:28] cond
[27:25] class (000=ALU reg, 001=ALU imm, 010=Mem, 011=Branch)
```

```
[24:21] opcode
[20]    S (update flags)
[19:16] Rn
[15:12] Rd
[11:0]  Op2/offset
```

Carte de Référence HDL

Structure d'un Fichier HDL

```vhdl
-- Déclaration de l'entité (interface)
entity NomComposant is
  port(
    entree1 : in bit;
    entree2 : in bits(7 downto 0);
    sortie  : out bit
  );
end entity;

-- Implémentation (architecture)
architecture rtl of NomComposant is
  -- Déclaration des composants utilisés
  component AutreComposant
    port(a : in bit; y : out bit);
  end component;

  -- Signaux internes
  signal sig1, sig2 : bit;
  signal bus1 : bits(7 downto 0);

begin
  -- Instanciation de composants
  u1: AutreComposant port map (a => entree1, y => sig1);

  -- Assignations concurrentes
  sortie <= sig1 and sig2;

end architecture;
```

Types de Données

Type	Description	Exemple
bit	Un seul bit (0 ou 1)	`signal x : bit;`
bits(n downto 0)	Vecteur de n+1 bits	`signal data : bits(31 downto 0);`

Opérateurs Logiques

Opérateur	Description	Exemple
`not`	Inversion	`y <= not a;`
`and`	ET logique	`y <= a and b;`
`or`	OU logique	`y <= a or b;`
`xor`	OU exclusif	`y <= a xor b;`
`nand`	NON-ET	`y <= a nand b;`
`nor`	NON-OU	`y <= a nor b;`

Accès aux Bits

```
signal data : bits(31 downto 0);

-- Bit individuel
data(0)           -- Bit de poids faible (LSB)
data(31)          -- Bit de poids fort (MSB)

-- Tranche (slice)
data(7 downto 0)  -- 8 bits de poids faible
data(31 downto 16) -- 16 bits de poids fort

-- Concaténation
result <= high_bits & low_bits;
```

Instanciation de Composants

```
-- Forme complète
u_and: And2 port map (
  a => signal_a,
  b => signal_b,
  y => signal_y
);

-- Plusieurs instances
u1: Not1 port map (a => in1, y => not_in1);
u2: Not1 port map (a => in2, y => not_in2);
```

Composants Primitifs

Composant	Ports	Description
Nand	a, b → y	Porte NAND (primitif)
DFF	d, clk → q	D Flip-Flop (primitif)

Composants de Base (à construire)

Composant	Ports	Description
Inv / Not1	a → y	Inverseur
And2	a, b → y	Porte AND
Or2	a, b → y	Porte OR
Xor2	a, b → y	Porte XOR
Mux	a, b, sel → y	Multiplexeur 2:1
DMux	in, sel → a, b	Démultiplexeur 1:2

Composants Arithmétiques

Composant	Ports	Description
HalfAdder	a, b → sum, carry	Demi-additionneur
FullAdder	a, b, cin → sum, cout	Additionneur complet
Add32	a[32], b[32] → sum[32], cout	Additionneur 32 bits
ALU32	a[32], b[32], op[4] → result[32], flags[4]	ALU complète

Composants Séquentiels

Composant	Ports	Description
Bit	in, load, clk → out	Registre 1 bit
Register	in[32], load, clk → out[32]	Registre 32 bits
PC	in[32], load, inc, reset, clk → out[32]	Program Counter
RAM8	in[32], addr[3], load, clk → out[32]	RAM 8 mots

Bonnes Pratiques

1. **Nommage** : Utilisez des noms descriptifs
   - `u_` pour les instances : `u_alu`, `u_mux`
   - `sig_` ou suffixe descriptif pour les signaux
2. **Organisation** :
   - Composants en premier
   - Signaux ensuite
   - Instanciations dans `begin...end`
3. **Commentaires** :
```
-- Description de ce que fait le bloc
u_add: Add32 port map (...);
```
4. **Hiérarchie** :
   - Construisez des composants simples d'abord
   - Réutilisez-les pour construire des composants complexes

Fichier de Test (.tst)

```
// Charger le composant
load MyComponent.hdl

// Définir les entrées
set a 0
set b 1

// Évaluer (circuits combinatoires)
eval

// Vérifier la sortie
expect y 1

// Pour circuits séquentiels
tick    // Front montant
tock    // Front descendant
step    // tick + tock
```

Carte de Référence C32

Types de Données

Type	Taille	Plage	Description
int	32 bits	-2^{31} à $2^{31}-1$	Entier signé
uint	32 bits	0 à $2^{32}-1$	Entier non-signé
char	8 bits	0 à 255	Caractère ASCII

Type	Taille	Plage	Description
bool	8 bits	true/false	Booléen
void	-	-	Absence de type
T*	32 bits	-	Pointeur vers T

Déclaration de Variables

```
int x;              // Non initialisé
int y = 42;         // Initialisé
int a, b, c;        // Multiples

int *ptr;           // Pointeur
int arr[10];        // Tableau de 10 éléments
```

Opérateurs

Arithmétiques

Op	Description	Exemple
+	Addition	`a + b`
-	Soustraction	`a - b`
*	Multiplication	`a * b`
/	Division	`a / b`
%	Modulo	`a % b`

Comparaison

Op	Description	Exemple
==	Égal	`a == b`
!=	Différent	`a != b`
<	Inférieur	`a < b`
>	Supérieur	`a > b`
<=	Inférieur ou égal	`a <= b`
>=	Supérieur ou égal	`a >= b`

Logiques

Op	Description	Exemple
&&	ET logique	`a && b`
||	OU logique	`a || b`
!	NON logique	`!a`

Bit à Bit

Op	Description	Exemple
&	ET	`a & b`
|	OU	`a | b`
^	XOR	`a ^ b`
~	NON	`~a`
«	Décalage gauche	`a << n`
»	Décalage droite	`a >> n`

Pointeurs

Op	Description	Exemple
&	Adresse de	`&x`
*	Déréférencement	`*ptr`

Structures de Contrôle

Conditionnelle

```
if (condition) {
    // si vrai
} else {
    // si faux
}
```

Boucle while

```
while (condition) {
    // corps
}
```

Boucle for

```
for (init; condition; increment) {
    // corps
}

// Exemple
for (int i = 0; i < 10; i = i + 1) {
    // répéter 10 fois
}
```

Boucle do-while

```
do {
    // corps (au moins 1 fois)
} while (condition);
```

Fonctions

```
// Déclaration
int add(int a, int b);

// Définition
int add(int a, int b) {
    return a + b;
}

// Appel
int result = add(5, 3);
```

Convention d'appel

- Arguments : R0, R1, R2, R3 (puis pile)
- Retour : R0
- Callee-saved : R4-R11

Pointeurs et Tableaux

```
int arr[5];           // Tableau
int *p = arr;         // Pointeur vers premier élément

arr[0] = 10;          // Accès par indice
*(arr + 1) = 20;      // Équivalent à arr[1]

p = p + 1;            // Avancer d'un élément
int val = *p;         // Lire la valeur pointée
```

Chaînes de Caractères

```
char *msg = "Hello";  // Chaîne constante

// Parcourir
while (*msg != '\0') {
    putc(*msg);
    msg = msg + 1;
}
```

Entrées/Sorties

Fonctions Intégrées

Fonction	Description
`putc(c)`	Afficher un caractère
`getc()`	Lire un caractère
`exit(n)`	Terminer avec code n

MMIO Direct

```
// Écran (framebuffer)
uint *screen = (uint*)0x00400000;
screen[0] = 0xFFFFFFFF;  // Ligne blanche

// Sortie caractère
char *output = (char*)0xFFFF0000;
*output = 'A';

// Entrée caractère
char *input = (char*)0xFFFF0004;
char c = *input;
```

Structures (si supporté)

```
struct Point {
    int x;
    int y;
};

struct Point p;
p.x = 10;
p.y = 20;
```

```
struct Point *ptr = &p;
ptr->x = 30;  // Équivalent à (*ptr).x
```

Bonnes Pratiques

1. **Toujours initialiser** les variables
2. **Vérifier les pointeurs** avant déréférencement
3. **Éviter la division par zéro**
4. **Utiliser des noms descriptifs**
5. **Commenter le code complexe**

Différences avec C Standard

Fonctionnalité	C32	C Standard
`i++`	Non	Oui
`float/double`	Non	Oui
`enum`	Non	Oui
`typedef`	Limité	Oui
Préprocesseur	Minimal	Complet
`malloc/free`	Via OS	stdlib

Carte de Référence - Codes d'Erreur

Erreurs d'Assemblage (E1xxx)

Code	Message	Cause	Solution
E1001	Unknown mnemonic	Instruction non reconnue	Vérifier l'orthographe
E1002	Invalid operand	Format d'opérande incorrect	Vérifier la syntaxe
E1003	Undefined label	Label non défini	Ajouter le label ou corriger le nom
E1004	Immediate out of range	Valeur trop grande	Utiliser `LDR Rd, =value`
E1005	Invalid register	Registre inexistant	Utiliser R0-R15
E1006	Syntax error	Erreur de syntaxe	Vérifier la ligne
E1007	Duplicate label	Label déjà défini	Renommer le label
E1008	Literal pool overflow	Trop de constantes	Ajouter `.ltorg`
E1009	Offset out of range	Offset trop grand	Utiliser registre intermédiaire
E1010	Invalid condition	Condition inconnue	Vérifier EQ, NE, GT, etc.

Exemples de Résolution

E1004 - Immediate out of range

```
; ERREUR
MOV R0, #0x12345678

; SOLUTION
LDR R0, =0x12345678
```

E1008 - Literal pool overflow

```
; ERREUR : trop de LDR Rd, =value sans .ltorg
LDR R0, =0x1000
; ... beaucoup de code ...
LDR R1, =0x2000  ; E1008!
```

```
; SOLUTION : placer .ltorg régulièrement
LDR R0, =0x1000
.ltorg
; ... code ...
LDR R1, =0x2000
.ltorg
```

Erreurs de Compilation C32 (E2xxx)

Code	Message	Cause	Solution
E2001	Parse error	Erreur de syntaxe	Vérifier parenthèses, ;
E2002	Type mismatch	Types incompatibles	Vérifier les types
E2003	Undeclared variable	Variable non déclarée	Déclarer avant usage
E2004	Undefined function	Fonction non définie	Définir la fonction
E2005	Redefinition	Double définition	Supprimer le doublon
E2006	Invalid lvalue	Assignation invalide	Vérifier le côté gauche
E2007	Argument count	Mauvais nombre d'args	Vérifier les paramètres
E2008	Missing return	Return manquant	Ajouter return
E2009	Break outside loop	Break hors boucle	Déplacer dans une boucle
E2010	Invalid cast	Cast impossible	Vérifier les types

Exemples de Résolution

E2003 - Undeclared variable

```
// ERREUR
int main() {
    x = 5;  // E2003
}

// SOLUTION
int main() {
    int x;
    x = 5;
}
```

E2008 - Missing return

```
// ERREUR
int add(int a, int b) {
    int c = a + b;
}  // E2008

// SOLUTION
int add(int a, int b) {
    int c = a + b;
    return c;
}
```

Erreurs de Liaison (E3xxx)

Code	Message	Cause	Solution
E3001	Unresolved symbol	Symbole non trouvé	Définir ou lier
E3002	Missing entry point	Pas de _start/main	Ajouter point d'entrée

Code	Message	Cause	Solution
E3003	Duplicate symbol	Symbole en double	Renommer
E3004	Memory overflow	Trop de données	Réduire la taille
E3005	Invalid section	Section inconnue	Vérifier .text/.data

Erreurs d'Exécution (Traps)

Trap	Cause	Diagnostic
DIV_ZERO	Division par zéro	Vérifier le diviseur
MEM_FAULT	Accès mémoire invalide	Vérifier le pointeur
MISALIGNED	Adresse non alignée	Aligner sur 4 octets
ILLEGAL	Instruction invalide	Vérifier l'encodage

Déboguer les Traps

DIV_ZERO

```
// ERREUR
int result = a / b;  // Si b == 0 → TRAP

// SOLUTION
int result = 0;
if (b != 0) {
    result = a / b;
}
```

MISALIGNED

```
; ERREUR
LDR R0, [R1, #3]  ; 3 n'est pas multiple de 4

; SOLUTION
LDR R0, [R1, #4]  ; ou #0, #8, #12...
```

MEM_FAULT

```
// ERREUR
int *p;           // Non initialisé
int x = *p;       // MEM_FAULT

// SOLUTION
int value = 42;
int *p = &value;
int x = *p;       // OK
```

Erreurs HDL

Erreur	Cause	Solution
Component not found	Composant inconnu	Vérifier le nom
Port not connected	Port non connecté	Ajouter port map
Width mismatch	Largeur incorrecte	Vérifier bit vs bits
Circular dependency	Dépendance circulaire	Utiliser DFF
Undefined signal	Signal non déclaré	Déclarer le signal

Conseils de Débogage

Par Type d'Erreur

Symptôme	Vérifier
Compilation échoue	Messages d'erreur, syntaxe
Résultat incorrect	Logique, conditions
Boucle infinie	Condition de sortie
Crash (trap)	Pointeurs, division
Valeur aléatoire	Initialisation

Checklist Rapide

- ☐ Variables initialisées ?
- ☐ Pointeurs valides ?
- ☐ Pas de division par zéro ?
- ☐ Return dans toutes les fonctions ?
- ☐ Labels correctement orthographiés ?
- ☐ Accès mémoire alignés ?
- ☐ LR sauvegardé avant BL ?