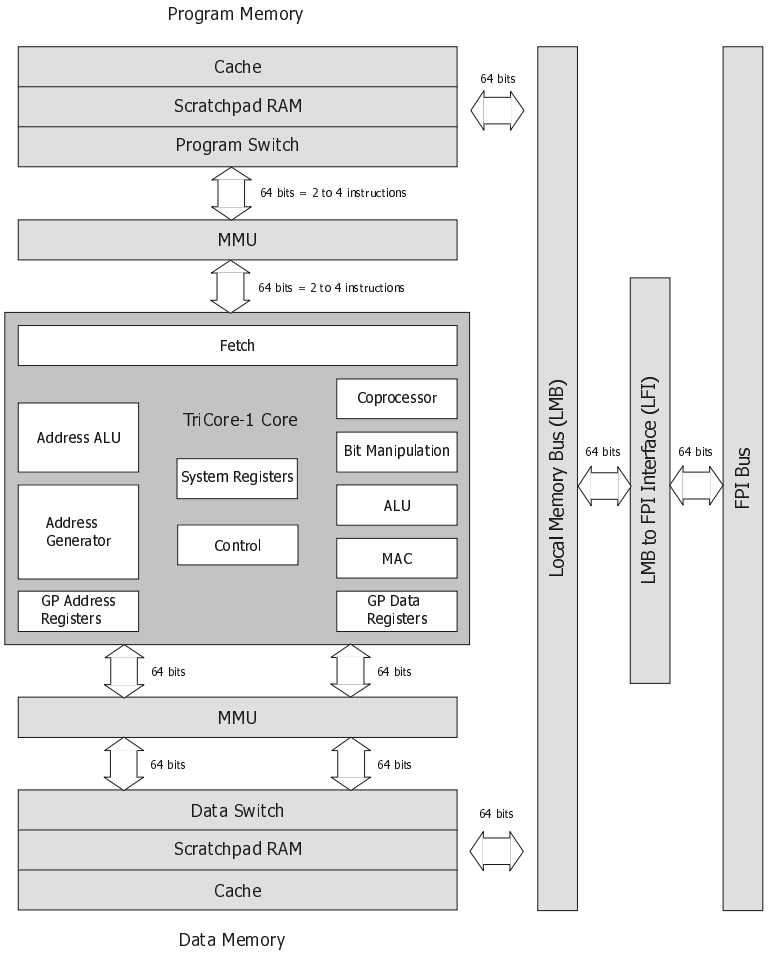
DSP/MCU: Infineon TriCore

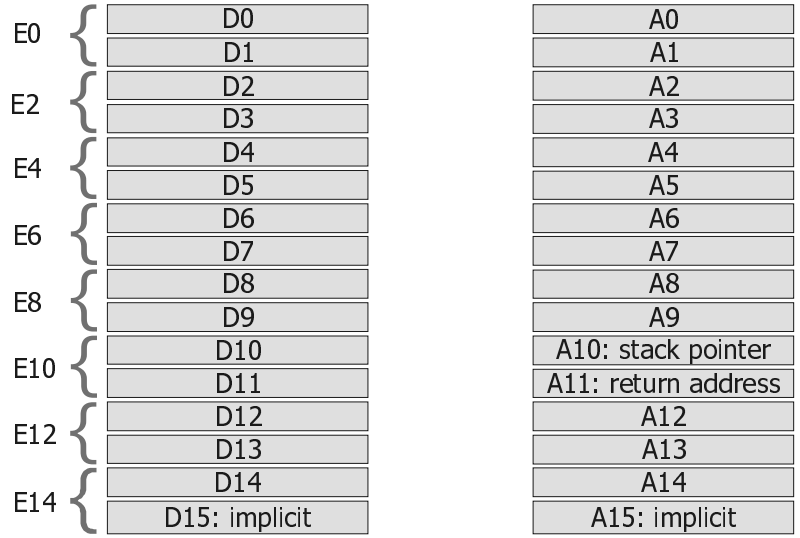


Infineon TriCore

- RISC-based 32-bit load/store architecture with Harvard architecture.
- 64-bit program memory bus between instruction fetch unit and program memory (fetch of up to 4 instructions).
- 128-bit data memory bus. Transfer of two data and two address registers in one clock cycle.

Infineon TriCore

- 32 general purpose registers separated in 16 address registers (A0-A15) and 16 data registers (D0-D15).



Infineon TriCore

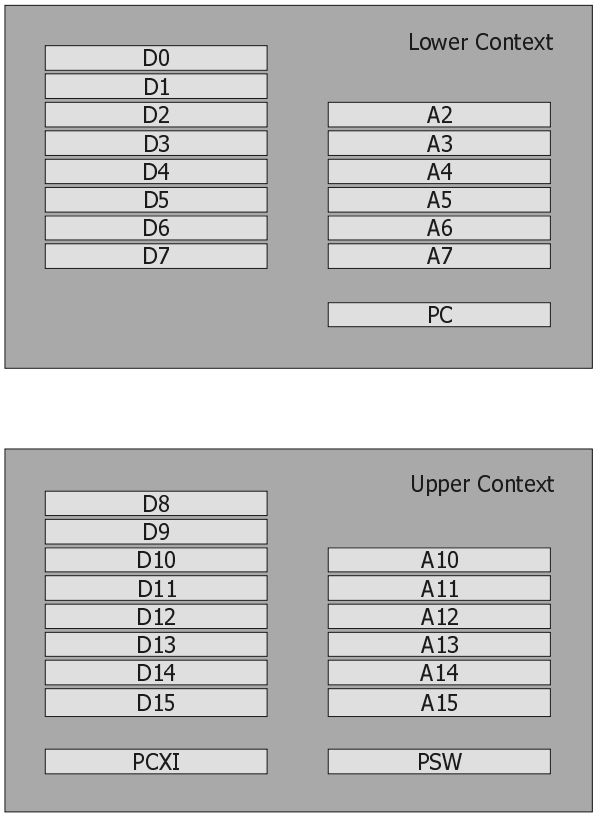
- The Core Special function registers (CSFRs) control the operation of the core and provide status information: program state information, context management, interrupt and trap control, etc.
- The CSFRs are mapped in the memory address space.
- Memory space:
  - address width: 32 bit
  - 4GB physical address space, divided into 16 sements of 256MB
  - 4GB virtual address space , divided into 16 sements of 256MB
  - First 16 KB of each segment can be accessed with absolute addressing

Infineon TriCore

Addressing Mode	Syntax	Effective Address
Absolute	const	constant
Base + short offset	[An]offset10	A[n] + sext(offset10)
Base + long offset	[An]offset16	A[n] + sext(offset16)
Pre-increment	[+An]offset10	A[n] + sext(offset10)
Post-increment	[An+]offset10	A[n]
Circular	[An+c]	A[n] + A[n+1][15:0]
Bit-reverse	[An+r]	A[n] + A[n+1][15:0]

Infineon TriCore

- Task: independent thread of control. Its state is defined by the task's context.
- Context switches occur on interrupts, traps and function calls.
- Upper context registers are automatically saved upon context switch in dedicated context save areas. They serve as non-volatile registers in function calling.
- Lower context registers can be used to pass parameters to functions.



Infineon TriCore

- Instructions are 32-bit wide; some instructions also have short 16-bit forms.
- 16-bit instructions use one of the following methods to limit the encoding width:
  - combined source and destination operands
  - implicit source, destination, or base address operand
  - small constants, short branch displacements
- Instructions for addition, subtraction, multiplication, multiply and accumulate, division, and shifting.

Infineon TriCore

- Bit field instructions, e.g. EXTR D1, D2, 3, 8
- Bit instructions, e.g. AND.T, and combined bit operations, e.g. AND.OR.T
- Control flow instructions: branches, loops, context save and restore, calls and returns.
- Zero-overhead loops:

```
.L23        mov.a a15, 149
            ld.h  d8,[a14+]
            add   d15,d8
            loop  a15, .L23
```

Infineon TriCore

- Branch prediction:
  - if the branch is a 16-bit instruction (backward or forward), the branch is predicted as taken.
  - if the branch is a 32-bit instruction and is a backward jump, it is predicted as taken.
  - if the branch is a 32-bit instruction and is a forward jump, it is predicted as not taken.
- Conditional instructions: SEL, CMOV, CADD, CSUB.

  Beispiel: D3 = ((d4 != 0) ? d1 : d2 implementiert durch SEL d3, d4, d1, d2

Infineon TriCore

- Fractional arithmetic:
  - Number *a=(*a_{m+n-1}*, …, *a_0*)* in *m*q*n*-format:

$$< a >= -a_{m+n-1}\,2^{m-1} + \sum_{i=0}^{m-2} a_{n+i}\,2^{i} + \sum_{i=0}^{n-1} a_i\,2^{i-n}$$

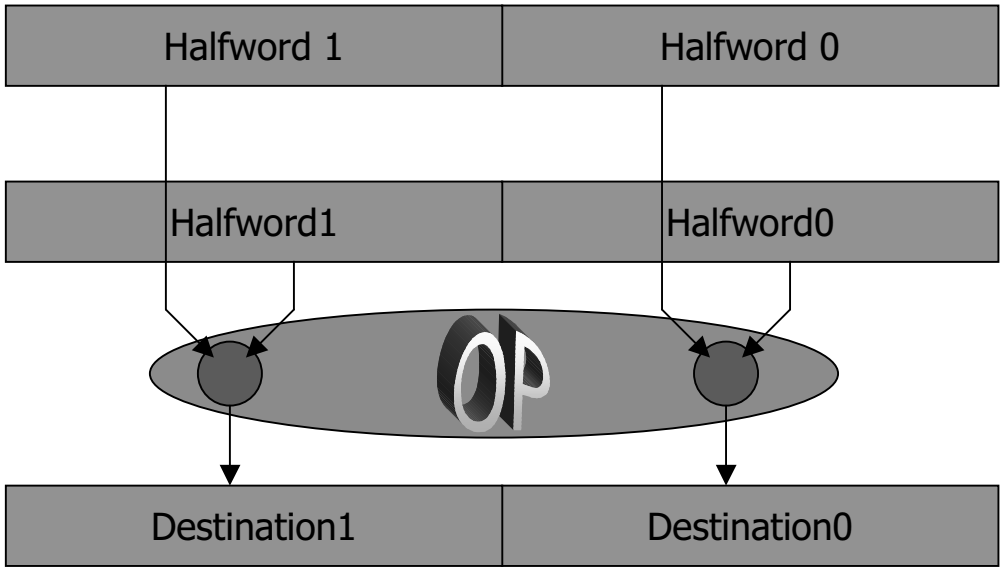
    Example: 2.14-format:

$$< a >= -a_{15}\,2^{1} + a_{14}\,2^{0} + a_{13}\,2^{-1} + \ldots + a_0\,2^{-14}$$

  - Frequently used: 1q31-format (Q31), 1q15 (Q15), 2q14 (Q14)
  - Addition of q-format numbers: input format must be identical; then output format is the same as input format.
  - Multiplication: *<a* q *b>* * *<c* q *d >* -> *<(a+c)>* q *<(b+d)>*
  - Characterized by the suffix .Q to the instruction mnemonic.

Infineon TriCore

- SIMD packed arithmetic:
  - Packed arithmetic instructions partition a 32-bit word into several identical objects.



Infineon TriCore

- SIMD packed arithmetic:
  - Packed arithmetic instructions partition a 32-bit word into several identical objects.
  - Two packed formats: 32-bit word divided into
    - two 16-bit halfword values (.H/.HU)
    - four 8-bit byte values (.B/.BU)
  - Load/store of packed values by normal load/store instructions.
  - Packed objects can be manipulated in parallel: addition, subtraction, multiplication, absolute value.
  - Example: ADD.H D1, D2, D3

Infineon TriCore

- Complex instructions, e.g. :

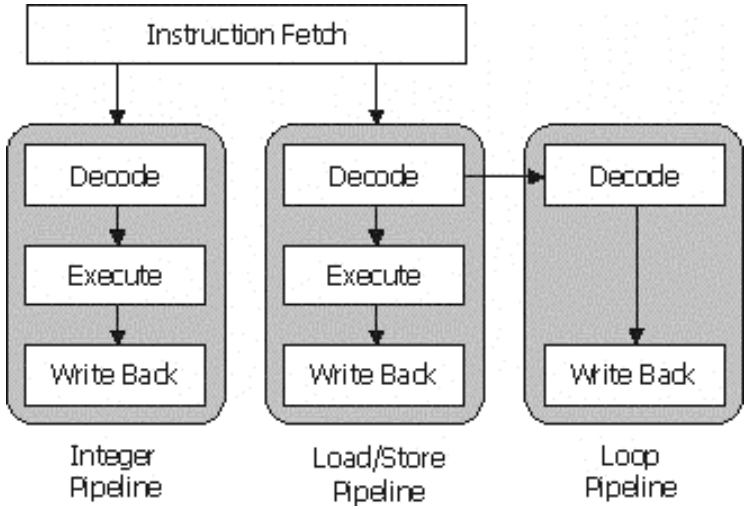
maddsum.h E2, E3, D0, D1UL, 1

⇔

E2[63:0] = E3[63:0] + (((D0[31:16] * D1[31:16])<<1)
- ((D0[15:0] * D1[15:0])<<1))

Infineon TriCore

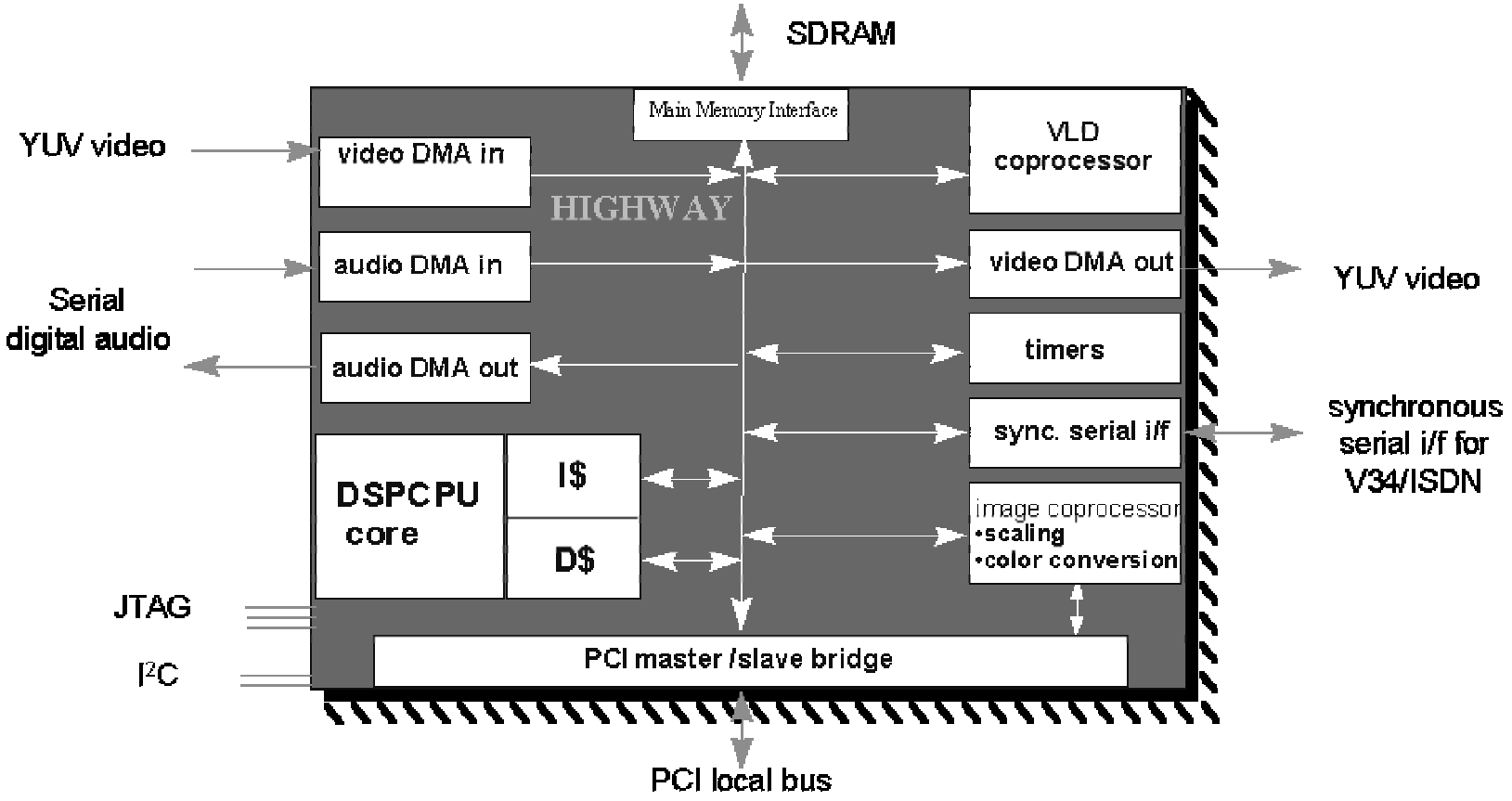
- Pipeline Mechanism: two major pipelines with four stages each (integer and load/store pipeline), and one minor pipeline for loop control.



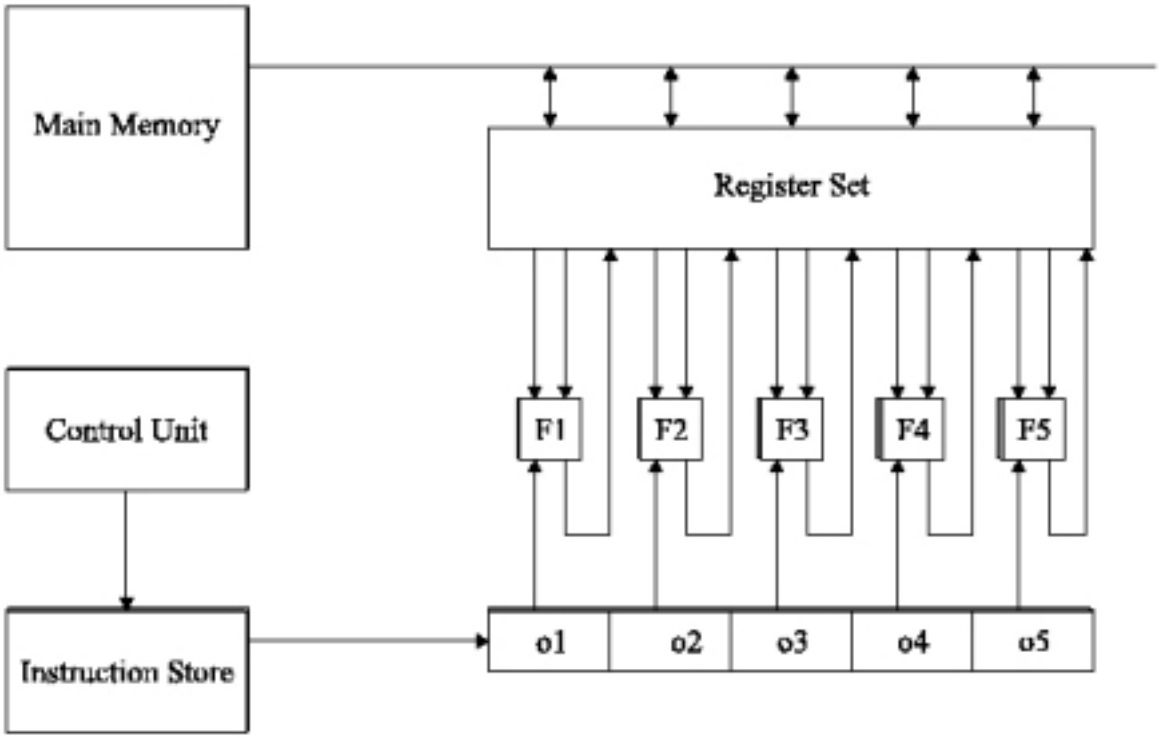
Infineon TriCore

- Instruction Timing:
  - If an LS instruction follows an IP instruction, both are executed in one clock cycle, except in case of MAC instructions and some *dual pipeline instructions*.
  - The execution time of instructions ranges from 1 to 4 clock cycles (best case assumption: no cache misses, no pending loads/stores).
  - LOOP/LOOPU instructions take one cycle on first execution of the loop and one cycle on loop exit; during the execution of the loop they do not take a cycle.
  - Branches:
    - correctly predicted as taken: 2 cycles
    - correctly predicted as not taken: 1 cycle
    - mispredicted: 3 cycles

VLIW: TriMedia TM1000 (1)



VLIW: TriMedia TM1000 (2)



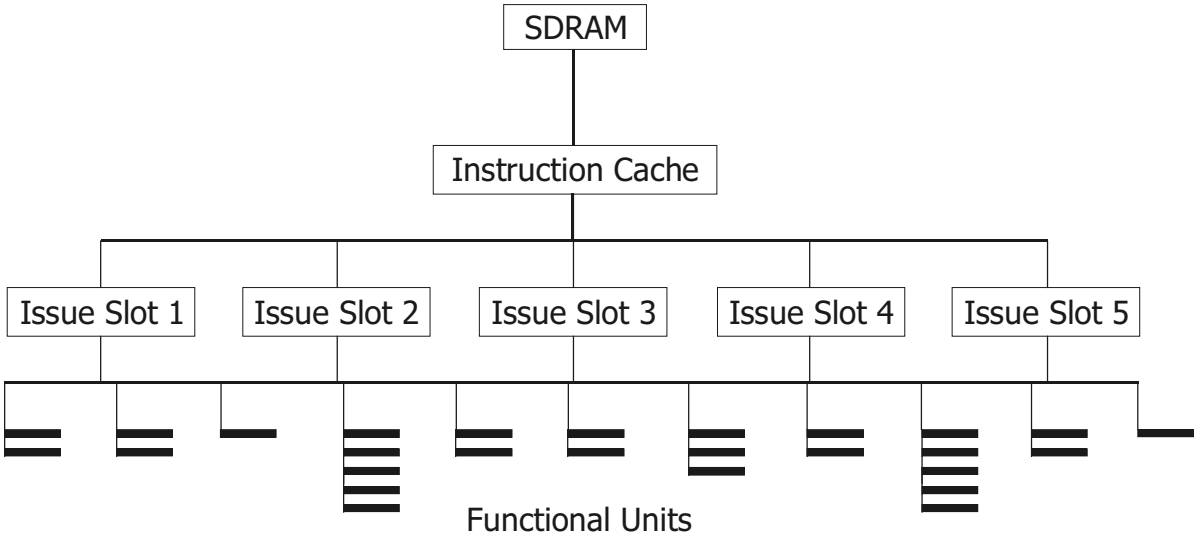
Generic VLIW Architecture

TriMedia TM1000

- Media processor for real-time processing of audio, video, graphics and communication.
- DSPCPU: 100 MHz VLIW core CPU; described in the following.
- 32-bit VLIW processor with 128 general purpose registers and 27 functional units.
- 16 KB dual-ported data cache and 32 KB instruction cache, both 8-way set-associative with 64 KB block size.
- Load/store architecture.

TriMedia TM1000

- Instruction set is composed of RISC-like operations, multimedia and DSP operations (SIMD; e.g. ifir)
- Each instruction word is composed of 5 microoperations.



TriMedia TM1000

- Execution time of operations is between 1 and 17 clock cycles.
- The packing of operations into instructions (assignment of operations to issue slots of the instruction word) is restricted:
  - E.g. no more than two load/store operations can be packed into a single instruction.
  - No more than 5 results can be written during any one cycle.
- Predicated execution: each operation can optionally be guarded. The operation is executed depending on the value of the general-purpose register specified as guard operand.

TriMedia TM1000

- Feasible issue slot combinations:

Type	Slot1	Slot2	Slot3	Slot4	Slot5
A	X	X	X	X	X
B				X	X
C	X	X			
D		X	X	X	
E	X		X		

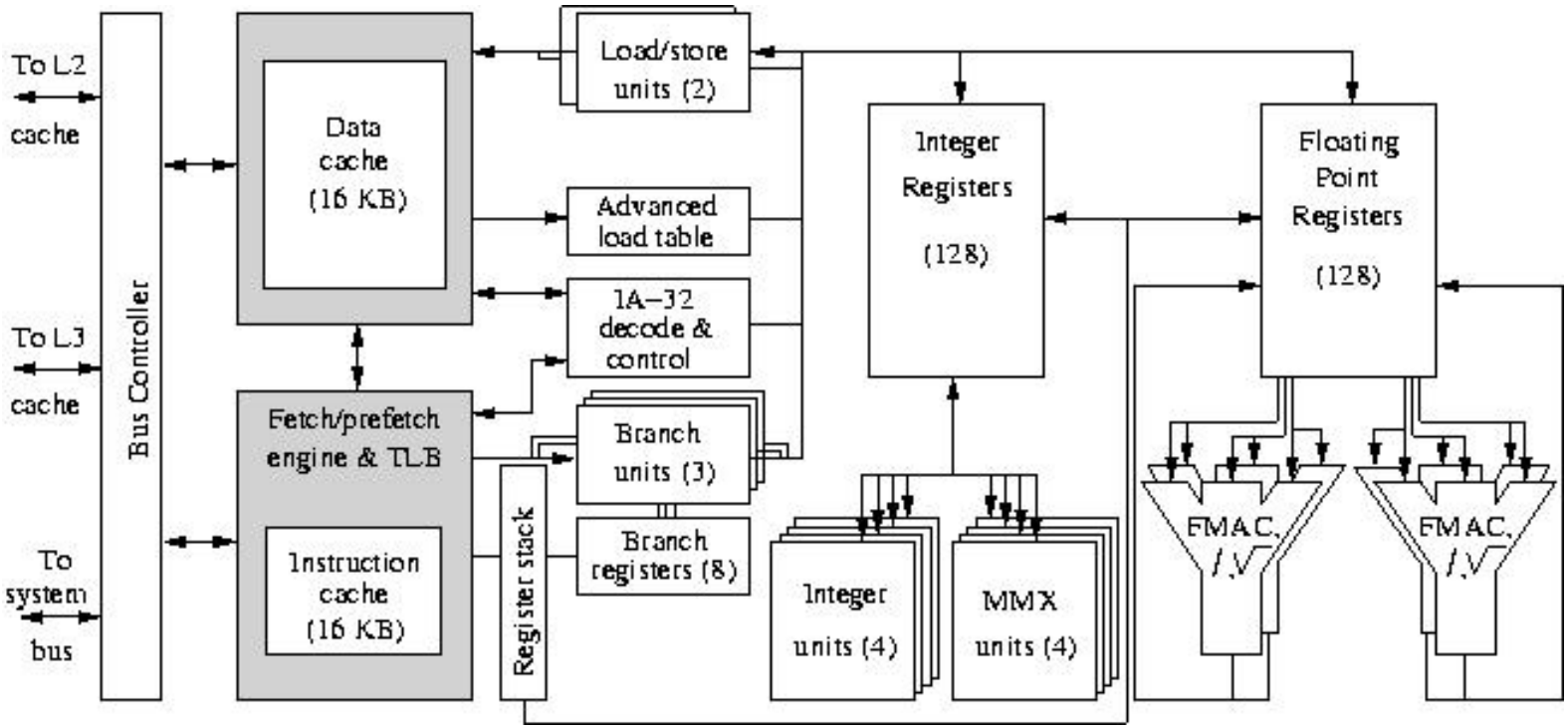
TriMedia TM1000

- Example code sequence:

```
(* cycle 0 *)
IF r1   iaddi(0x2) r0 -> r38,   IF r1   isubi(0x4) r0 -> r7,
IF r1   isubi(0x3) r0 -> r8,    IF r1   isubi(0x2) r0 -> r36,
IF r1   isubi(0x1) r0 -> r37;

(* cycle 1 *)
IF r1   iaddi(0x3) r0 -> r39,   IF r1   uimm(0x190) -> r34,
IF r1   iaddi(0x8) r0 -> r35,   IF r1   h_st8d(0) r7 r6,
IF r1   h_st8d(4) r0 r6;
```

EPIC: IA64 - Intel Itanium



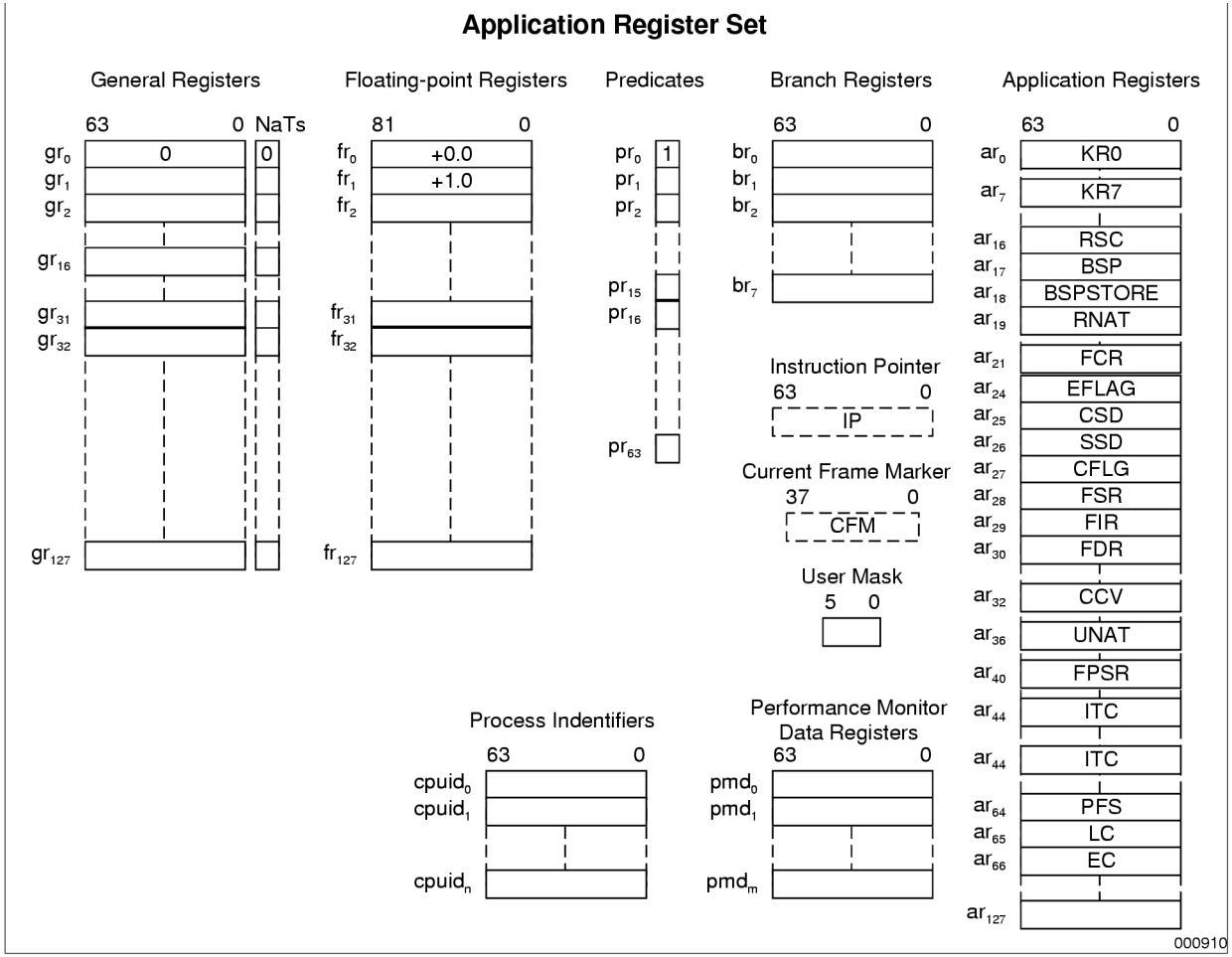
Block diagram of Intel Intanium.

IA-64 Architecture

- Support for
  - branch prediction (minimize the cost of branches),
  - predicated execution (remove branches),
  - control speculation and data speculation (reduce memory latency effects)
  - cache hints (compiler prediction of spatial or temporal locality of the memory area being accessed; can be used for placement of cache lines in the cache hierarchy by the processor)

- Register sets:
  - 128 64-bit general purpose registers: GR0-GR127
  - 128 80-bit floating-point registers: FR0-FR127
  - 64 1-bit predicate registers: PR0-PR63
  - 8 64-bit branch registers: BR0-BR7
  - features: register stack and rotating registers

IA-64 Architecture

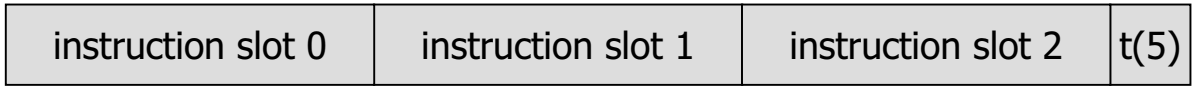


IA-64 Architecture

- Each IA-64 instruction is categorized into 6 types and may be executed on one or more execution unit types.
- 4 functional unit categories:
  - I unit (integer)
  - F unit (floating-point)
  - M unit (memory)
  - B unit (branch)
- 6 microoperation categories:
  - Integer ALU (A-type) — executed on M- or I units
  - Non-ALU Integer (I-type) — executed on I units
  - Memory (M-type) — executed on M units
  - Floating-point (F-type) — executed on F units
  - Branch (B-type) — executed on B units
  - Extended (L/X-type) — executed on I- or B units

IA-64 Architecture

- The IA-64 defines 128-bit long instruction words (called bundles) consisting of three 41-bit microoperations and a 5-bit template field. Multiple bundles can be issued per clock cycle (number is defined by implementation).



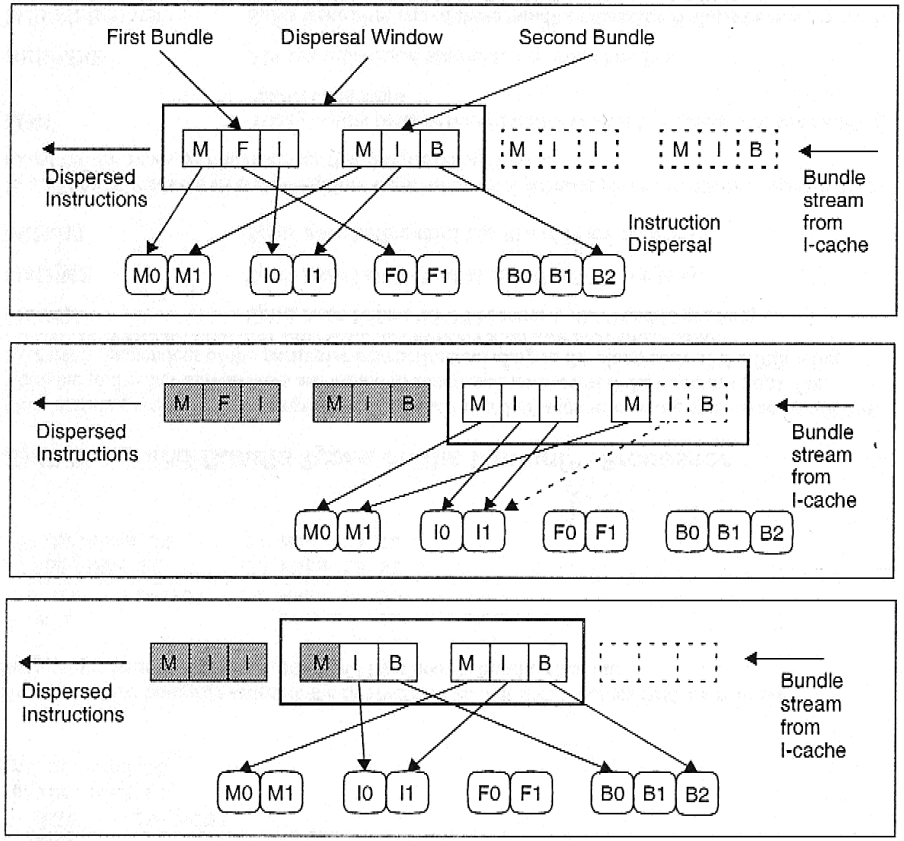
- Template field:
  - helps decode and route instructions
  - indicates the location of stops that mark the end of groups of microoperations that can execute in parallel

- Dispersal: process of sending instructions to functional units.

IA-64 Architecture

- There are 12 basic bundle types: MII, MI_I, MLX, MMI, M_MI, MFI, MMF, MIB, MBB, BBB, MMB, MFB. Each basic type has two versions, one with a stop after the third slot and one without.
- An instruction group is a sequence of instructions starting at a given bundle address up to the first stop, taken branch, break.b instruction fault or illegal operation fault.
- Instruction groups can extend over an arbitrary number of bundles and are determined by the location of stops in assembly code.
- Instruction execution phases:
  - fetch: read the instruction from memory
  - read: read architectural state
  - execute: perform the specified operation
  - update: update architectural state, if necessary

IA-64 Architecture: Bundling and Dispersal on Itanium



IA-64 Architecture

- Between instruction groups: instructions behave as though read accesses occur after the update of all instructions from the previous group.

- Within instruction groups: instructions behave as though
  - memory read accesses occurred after the update of the memory of all prior instructions in the same group.
  - register read accesses occurred before the update of the register state by any instruction (prior or later) in the same instruction group – with the exception of some dependence requirements (see next slide).

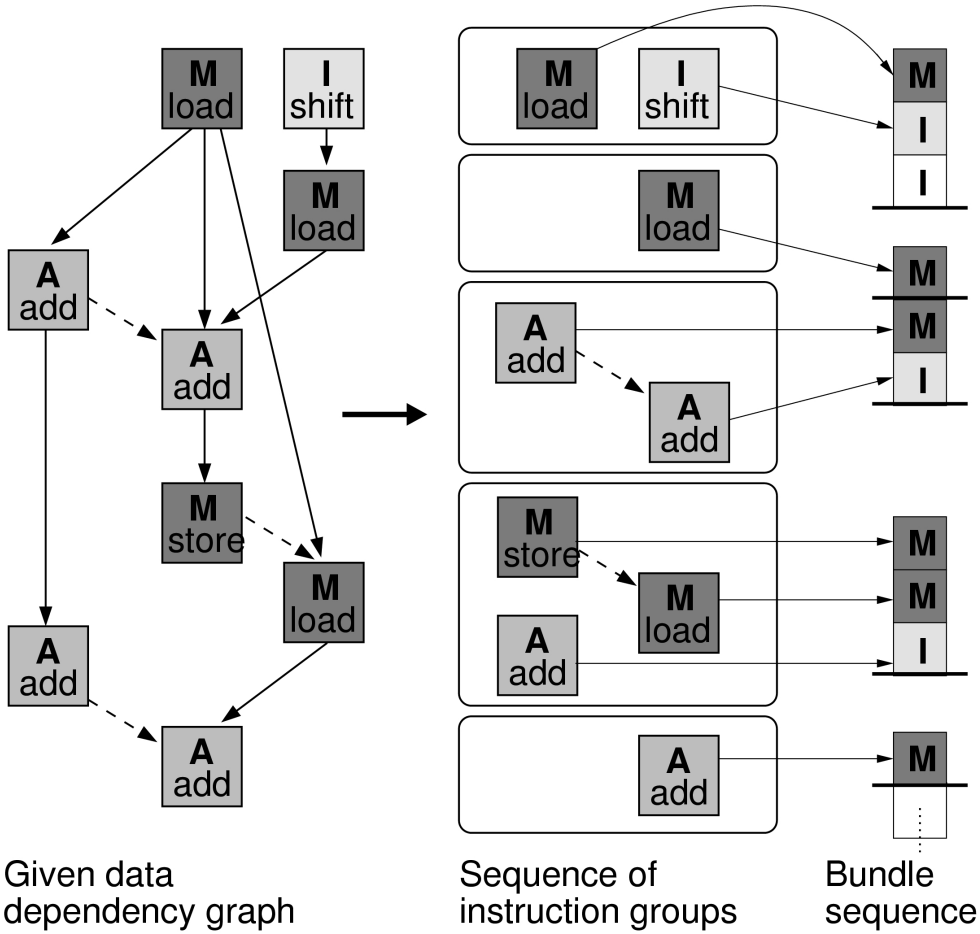
IA-64 Architecture

- Register dependencies: Within an instruction group, read-after-write (RAW) and write-after-read (WAR) register dependencies are not allowed (with respect to explicit and implicit register accesses), except for some special cases. Write-after-read register dependencies (WAR) are allowed, except for some special cases.

- If the program violates the dependency rules within an instruction group, processor behavior is undefined.

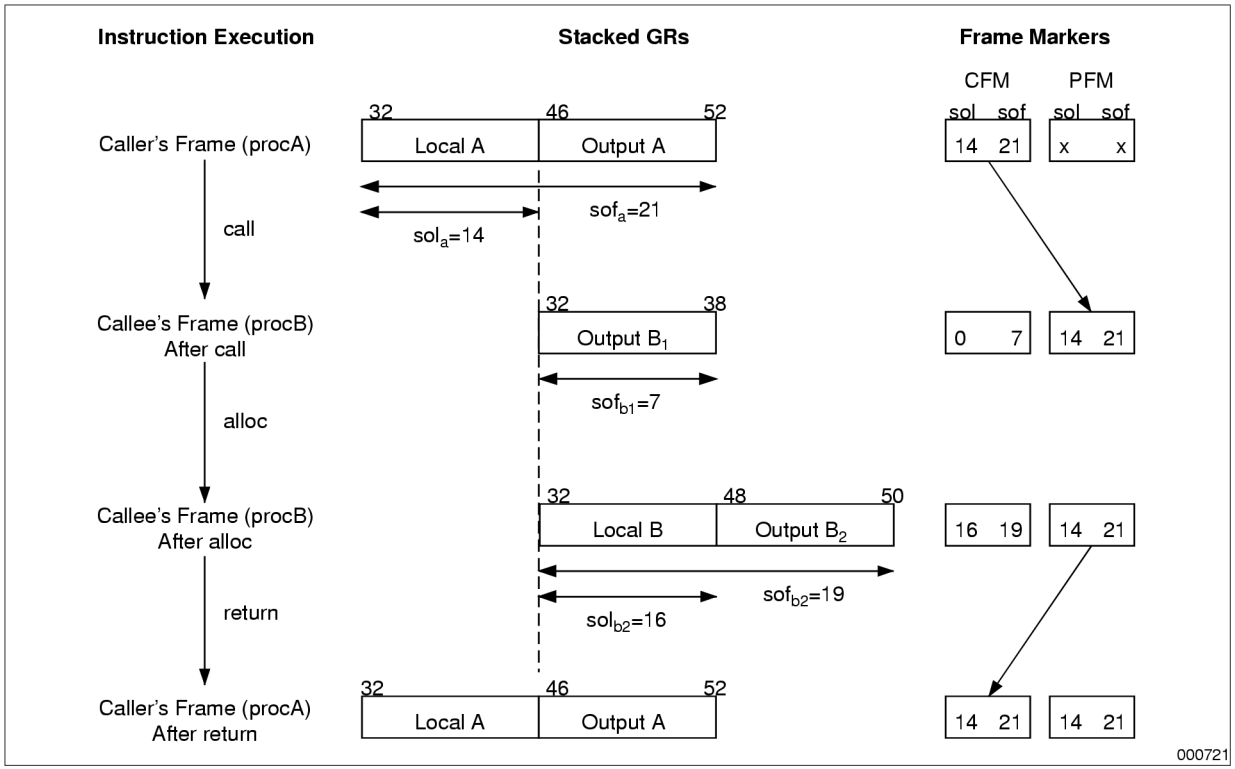
- Memory dependencies: Within an instruction group, RAW, WAW, and WAR memory dependencies are allowed. A load will observe the results of the most recent store to the same memory address.

IA-64 Architecture



IA-64 Architecture

- Register Stack Behavior on Procedure Call and Return



IA-64 Architecture

- Compare instructions and predication:
  - A compare instruction tests for a single specified condition and generates a boolean result which is written to predicate registers. The predicate registers can be used as conditions for conditional branches and as qualifying predicates for predication.
  - Most compare instructions set two predicate registers.
  - Compare instructions can have 5 types: Normal, Unconditional, AND, OR, or DeMorgan. The type defines how the instruction writes its target predicate registers.
  - AND, OR, and DeMorgan types are termed "parallel" compare types since they allow multiple simultaneous compares to target a single predicate register.

IA-64 Architecture

- Compare Types:

Compare Type	Completer	Operation	
		First Predicate Target	Second Predicate Target
Normal	*none*	if (qp) {target = result}	if (qp) {target = !result}
Unconditional	unc	if (qp) {target = result} else {target = 0}	if (qp) {target = !result} else {target = 0}
AND	and	if (qp && !result) {target = 0}	if (qp && !result) {target = 0}
	andcm	if (qp && result) {target = 0}	if (qp && result) {target = 0}
OR	or	if (qp && result) {target = 1}	if (qp && result) {target = 1}
	orcm	if (qp && !result) {target = 1}	if (qp && !result) {target = 1}
DeMorgan	or.andcm	if (qp && result) {target = 1}	if (qp && result) {target = 0}
	and.orcm	if (qp && !result) {target = 0}	if (qp && !result) {target = 1}

IA-64 Architecture

- Parallel compare instructions:

```
if (r1==1 || r2==1 || r3==1 || r4==1)
then s;

cmp.ne p1,p0 = r0, r0;;   //init
cmp.eq.or p1,p0 = 1,r1
cmp.eq.or p1,p0 = 1,r2
cmp.eq.or p1,p0 = 1,r3
cmp.eq.or p1,p0 = 1,r4;;
(p1) s
```

IA-64 Architecture

- Speculation: premature execution of an operation
  - conditions:
    - low probability of required recovery
    - enhancement of instruction-level parallelism by speculative execution.
- Control speculation: Execution of an operation before the branch which guards it.
- Data speculation: Execution of a memory load prior to a store that precedes it in the operation sequence and that may alias with it.

IA-64 Architecture

Control speculation:

```
(p1)    br.cond label
        ld8 r4=[r3];;
        add r4=8,r4;;
```



```
        ld8.s r4=[r3];;
// ...
(p1)    br.cond label;;
        chk.s r4, recovery;;
back:   add r4=8,r4;;
// ...
recovery:
        ld8 r4=[r3];;
        br back;;
```

IA-64 Architecture

- Data Speculation:

```
//...
st8 [r4]=r12
ld8 r6=[r8];;
add r5=r6,r7;;
st8 [r18]=r5
```

```
        ld8.a r6=[r8];;
        // ...
        add r5=r6,r7;;
        // ...
        st8 [r4]=r12
        chk.a.clr r6, recovery
back:   st8 [r18]=r5
        // ...
recovery:
        ld8 r6=[r8];;
        add r5=r6,r7
        br back;;
```