6617: Arquitectura RISC de ancho de palabra de datos parametrizable para implementaciones sobre tecnología FPGA

Julián U. da Silva Gillig, Miguel Ángel Sagreras, Alberto Dams
jdasilva@dc.uba.ar, msagre@cactus.fi.uba.ar, adams@cactus.fi.uba.ar

Laboratorio de Sistemas Digitales, Departamento de Electrónica, Facultad de Ingeniería,
Universidad de Buenos Aires
Av. Paseo Colón 850, (1063) Ciudad de Buenos Aires
Tel.: 4342-9184, int. 273.

Resumen

En este trabajo se propone la arquitectura básica de un microprocesador de tipo RISC, de ancho de palabra de datos parametrizable, diseñado para ser implementado sobre tecnología FPGA (Field Programmable Gate Array). El mismo está orientado especialmente a aplicaciones en sistemas embebidos, por lo que se pretende optimizar el uso de memoria de programa, manteniendo además un buen desempeño temporal de ejecución. Se hace hincapié en la reducción de dependencias entre instrucciones y otros aspectos que facilitan las optimizaciones de compilación y la generación de código eficiente temporal y espacialmente. Otro objetivo importante del diseño es reducir su complejidad, sobre todo teniendo en cuenta el tipo de tecnología de implementación a utilizar.

1. Introducción

Las condiciones de diseño están dadas principalmente por los siguientes factores:

- El tipo de aplicaciones a las que está orientado primariamente (sistemas embebidos, aplicaciones de control, computación portátil de baja potencia).
- La tecnología de implementación (FPGA –*Field Programmable Gate Array*-).
- La información estadística de uso de instrucciones y desempeño en procesadores reales [1].

A continuación se mencionan las principales condiciones de diseño:
1) Instrucciones de largo fijo de 16 bits.
2) Tamaño de palabra de datos parametrizable (16-32 bits).
3) 16 registros (como mínimo) de uso general ortogonales más el contador de programa.
4) Conjunto de instrucciones reducido (libre de redundancias, en lo posible) y donde se busca una alta simetría y ortogonalidad a nivel instrucción (además de la ya mencionada a nivel registro).
5) Tamaño fijo del campo de bits que identifica a cada instrucción (*opcode*).
6) Arquitectura *load/store* estricta (taxonomía *<0, x>* según [1, pp.70-73], donde *0* es la cantidad de operandos en memoria y *x* la cantidad de operandos en registros internos).
7) Carga inmediata de constantes de 8 bits.

2. Aspectos básicos de la arquitectura

Los formatos de instrucción, el sistema de registros y los modos de direccionamiento son los primeros elementos a definir, por la fuerte relación existente entre ellos.

2.1. Registros

El procesador cuenta con 16 registros visibles de uso general (R0-R15). No se mencionarán en este trabajo los registros ocultos que puedan aparecer en la implementación, como los que almacenan valores temporales entre las etapas de *pipeline*. No hay restricciones en cuanto al uso de los registros generales. Esto significa que todos los modos de direccionamiento y todas las operaciones que involucran registros pueden utilizarlos de forma indistinta e independiente. La única salvedad es el uso de R0 para las direcciones de retorno de las instrucciones JAL y JRAL (*Jump And Link* y *Jump Register And Link*). El contador de programa (PC) está separado de estos registros. No hay registro de estado (ver [2], por ejemplo), ya que esto genera dependencias entre instrucciones. En su lugar, cada registro posee 2 bits asociados: T (*Test*) y C (*Carry*). En la sección 3 (saltos condicionales) se profundiza este punto. La figura 1 muestra la estructura de cada registro (con el procesador configurado en 16 bits).

Bits	***-***		***15***	***14-0***
Designación	T	C	(N)	-

Fig. 1. Registros de uso general y sus bits de estado.

El bit más significativo es considerado como el flag N (*Negative*). En la figura 2 se ve un esquema de todos los registros visibles.

PC

T	C	(N)	***R0***
			...
T	C	(N)	***R15***

Fig. 2. Registros visibles del procesador.

2.2. Formatos de instrucción.

La tabla 1 muestra los 3 formatos de instrucción: I, J y R. El código de operación será de 4 bits, obteniéndose un espacio de 16 instrucciones. Su posición es fija, así como la de los demás campos. Esto simplifica la lógica de decodificación y la estructura interna del procesador.

Tabla 1. Formato de las instrucciones.

Tipo	*Campos (Cantidad de bits)*			
J	*Op (4)*	*Imm (12)*		
I	*Op (4)*	*Rd (4)*	*Imm (8)*	
R	*Op (4)*	*Rd (4)*	*Rs/Imm (4)*	*Imm/Func (4)*

Todas las operaciones aritmético-lógicas entre registros son del tipo R. Por lo tanto, la cantidad de operandos es 2. Dichos operandos están representados en campos de de 4 bits, para direccionar a los 16 registros. El destino de la operación está implícito, siendo uno de los operandos (Rd en la figura). Las condiciones de diseño 1, 3 y 5 quedan satisfechas con esta elección de formatos. La carga de constantes inmediatas de 8 bits (condición 7) se hará con una instrucción del tipo I. El tipo J permitirá hacer saltos incondicionales relativos al PC con un desplazamiento de 12 bits. También serán de este tipo las instrucciones de coprocesador, o excepciones por *software* (COP en este conjunto de instrucciones, TRAP en otros procesadores. Ver [1], [2], [3]).

2.3. Modos de direccionamiento

Los modos que posee son *inmediato*, *directo* e *indirecto con desplazamiento* (instrucciones LW/SW, LH/SH y SB) No existen más operaciones sobre la memoria que la carga o el almacenamiento de datos (condición 6). Los movimientos entre registros son realizados por la instrucción aritmética MOV (tipo R). Todos los registros sirven como punteros en el direccionamiento indirecto (condiciones 3 y 4). Las operaciones relacionadas con la unidad aritmético lógica (ALU) trabajan con los modos inmediato y directo (si bien en el modo inmediato se hacen algunas salvedades, se guarda una alta simetría, como se podrá ver en la sección 5).

3. El sistema de saltos condicionales

Un punto que se encaró de forma distinta a la mayoría de los procesadores comerciales es el de los saltos condicionales y los bits de estado o condición. Es conocido el problema de generación de dependencias ([1, pp.80-85]) cuando el procesador cuenta con un registro de estado. Esto obliga a seguir un orden determinado en la ejecución de instrucciones, lo que dificulta la optimización del código (la cual ha sido priorizada en este diseño). En procesadores como el DLX (ver [1, p.104]), el problema se resuelve con instrucciones que colocan en 0 un registro dependiendo de la condición a evaluar (*set conditionals*), para que luego los saltos condicionales decidan en base a si dicho registro es igual o distinto de 0 (instrucciones del tipo BRZ y BRNZ). Esto puede no tener un impacto negativo apreciable cuando se cuenta con 32 o más registros generales, pero en procesadores más pequeños el problema no es menor. Instrucciones de ese tipo destruyen el contenido del registro de destino, lo que acrecienta la cantidad de operaciones necesarias. En el 6617 se optó por colocar los bits de estado asociados a cada registro (figura 1). Esto permite obtener un eficiente sistema de saltos condicionales, con sólo 2 instrucciones (la escritura condicional del bit y el salto condicional consiguiente), sin perder contenidos ni forzar el orden de ejecución. Sin embargo, esta solución acarreó una complicación: la evaluación contra el literal 0 tras las operaciones de ALU (caso muy común –ciclos tipo *for*, por ejemplo-) requería del agregado de la instrucción de escritura condicional del bit. Por esto, se definió que todas las operaciones de ALU (tanto inmediatas como directas) modificaran al bit T de la misma forma en que lo harían si existiera el código de condición Z (*Zero*).

Por último, unas palabras sobre la instrucción de salto condicional propuesta (BRT): el campo de desplazamiento es de 8 bits (instrucción tipo I). Según las pruebas estadísticas de ejecución de [1, pp.82-84], cerca del 90% de los saltos condicionales están dentro de este rango.

4. Parametrización del ancho de palabra de datos

Como se estableció en la condición 2, el tamaño de la palabra de datos es parametrizable. Esto se pensó sobre todo para que el procesador sea configurado en los modos de 16 ó 32 bits (si bien otros valores son posibles, no se ha hecho un estudio aún). Para el diseño de la arquitectura se estableció que el código de 16 bits pueda correr sin cambios en procesadores configurados en modo 32 bits. A la inversa no se asegura compatibilidad. Las únicas instrucciones no presentes en el modo 16 son LW y SW, las cuales cargan y almacenan palabras de 32 bits entre la memoria y los registros. La aparición de sus *opcodes* en un procesador de 16 bits generará una excepción interna. Una de las ventajas principales de esta arquitectura reside justamente en que puede mantener un largo de programa reducido (debido a las instrucciones de 16 bits) y trabajar con datos de 32 bits y una capacidad de direccionamiento de 4 Giga bytes.

5. Resumen del conjunto de instrucciones

5.1. Instrucciones

En la tabla 2 se resume el conjunto completo de instrucciones del procesador propuesto, con el formato (tipo) al que pertenecen.

Tabla 2. Conjunto de instrucciones completo.

Transferencia de datos		
Instr.	***Tipo***	***Designación***
LW	R	Load Word (sólo modo 32 bits)
SW	R	Store Word (sólo modo 32 bits)
LH	R	Load Half word
SH	R	Store Half word
SB	R	Store Byte
LIH	I	Load Immediate High
MOV	R	MOVe Data from register to register
Control de programa		
Instr.	***Tipo***	***Designación***
COP	J	COProcessor
JAL	J	Jump And Link
JRAL	I	Jump Register And Link
BRT	I	Branch if register's Test bit is set
Comparación		
Instr.	***Tipo***	***Designación***
TEQ	R	set Test bit if Equals
TNE	R	set Test bit if Not Equals
TLT	R	set Test bit if Lower Than
TGT	R	set Test bit if Greater Than
TLE	R	set Test bit if Lower or Equal than
TGE	R	set Test bit if Greater or Equal than
TEQC	R	set Test bit if Equals (with Carry)
TNEC	R	set Test bit if Not Equals (with Carry)
TLTC	R	set Test bit if Lower Than (with Carry)
TGTC	R	set Test bit if Greater Than (with Carry)
TLEC	R	set Test bit if Lower or Equal than (with Carry)
TGEC	R	set Test bit if Greater or Equal than (with Carry)
TCF	R	set Test bit if Carry Flag is set
TNCF	R	set Test bit if No Carry Flag is set
TNF	R	set Test bit if Negative Flag is set
TNNF	R	set Test bit if No Negative Flag is set
Aritméticas / Lógicas		
Instr.	***Tipo***	***Designación***
ADD	R	ADD registers
ADDC	R	ADD Carry to register
ADDI	I	ADD Immediate to register
SUB	R	SUBstract registers
SUBC	R	SUBstract Carry from register
CPI	I	ComPare Immediate
ASR	R	Arithmetical Shift Right register
ASRI	R	Arithmetical Shift Right register Immediate
LRS	R	Logical Right Shift register
LLS	R	Logical Left Shift register
LLSI	R	Logical Left Shift register Immediate
LRSI	R	Logical Right Shift register Immediate
AND	R	logical AND registers
ANDI	I	logical AND register with Immediate
OR	R	logical OR registers
ORI	I	logical OR register with Immediate
XOR	R	logical XOR registers
NOR	R	logical NOR registers
NEG	R	2's complement NEGate register

5.2. Pseudoinstrucciones

En la tabla 3 se ven algunas instrucciones que en realidad son mapeos realizados por el ensamblador sobre el conjunto real del procesador.

Tabla 3. Algunas pseudoinstrucciones posibles.

Instr.	***Sintaxis***	***Instr. real***	***Designación***
NOT	NOT Ri	NOR Ri, Ri	logical NOT register
CLR	CLR Ri	XOR Ri, Ri	CLear Register
NOP	NOP	AND Ri, Ri	No OPeration
RFE	RFE	COP K12	Return From Exception
SUBI	SUBI Ri, K8	ADDI Ri, -K8	SUBstract Immediate

Ri representa a cualquier registro general (R0-R16); K8 y K12 son valores inmediatos de 8 y 12 bits, respectivamente.

6. Conclusiones y estado del desarrollo

Es cierto que el verdadero valor de una arquitectura recién se conoce al ser sometida al análisis estadístico sobre una implementación (ya sea de *hardware* o simulada) del procesador en cuestión. Las estimaciones que se realizaron, indican que se generará aproximadamente un 20 % más de instrucciones que las presentadas en las estadísticas de [1]. Cabe destacar que las instrucciones aquí utilizadas son de 16 bits, contra 32 de [1].
Al momento de escribir este artículo se terminó de portar el compilador de GNU (gcc) para el 6617. Se está trabajando además sobre un simulador. Con la finalización del compilador compenzó también la migración del kernel de Linux sobre esta arquitectura.
Se espera de este modo que las propuestas aquí consignadas y las herramientas desarrolladas puedan servir como base para avanzar en el desarrollo de un procesador con instrucciones de 16 bits y palabras de datos parametrizables (16-32) donde pronto podrán realizarse pruebas de ejecución. Creemos que especialmente interesante puede resultar la solución planteada para el sistema de saltos condicionales, muy ligado a los *flags* de estado por registro.
Otro punto en el que se hizo especial hincapié fue en el de mantener la ortogonalidad y la simpleza del procesador, de modo que no se recurrió a "desprolijidades", muy usadas incluso en procesadores que comercialememte sus fabricantes denominan RISC. Ejemplos de estas prácticas son *opcodes* e incluso instrucciones de largo variable, registros especiales para carga de constantes o para mantener *flags* de estado, etc.. Todo esto suele redundar en la aparición de dependencias (dificultando la optimización del código) o en ciclos de ejecución por instrucción más largos, lo cual además trae aparejada una falta de balance que complica la lógica de *pipelining* o incluso agrega tiempos muertos. También pueden complicar la lógica. La implementación del 6617 funcionando con palabras de datos de 16 bits y *pipeline* de 5 etapas requiere 300 elementos lógicos en una ACEX1K de Altera, la cual puede operar de 0 a 100 MHz con el dispositivo más rápido y de 0 a 30 MHz con el más lento de la familia. Una ventaja adicional de disponer de procesadores sobre FPGAs es la posibilidad de incorporar periféricos a medida de la aplicación, o incluso más de un procesador corriendo en paralelo o trabajando sobre funciones dedicadas.

7. Referencias

[1] Patterson, D. A., Hannessy, J. L., con contribución de D. Goldberg. *Computer Architecture: a quantitative approach*, Morgan Kaufmann Publishers, San Francisco, U.S.A., 1996.

[2] Altera Corporation. *Nios Embedded Processor: Programmer's Reference Manual*, Altera Corporation, U.S.A., Julio de 2001.

[3] Atmel Corporation, con acuerdo de licencia de Advanced RISC Machines Limited, *ARM7TDMI™: (Thumb®) Datasheet*, Atmel ES2, Francia, Enero de 1999.