Desarrollo de las ideas principales para el desarrollo

Información General Del Desarrollo Del Proyecto

Instrucciones Generales:

1. Se intentaran tener instrucciones de un tamaño constante para recudir la complejidad del sistema de decodificación. Y el tamaño será de **64bits**.
2. Nos basaremos en el conjunto de instrucciones de ARMv4 para las instrucciones generales del sistema computacional, pero, para las instrucciones complejas de nuestra implementación computacional intentaremos utilizar algo mas especifico.
3. La microarquitectura por utilizar va a ser **Pipeline**, con una implementación de Hazard utilizando la identificación de stalls necesarios, y así evitar los problemas de RAW, WAR, WAW. Aparte se tomarán en cuenta las condiciones estructurales.
4. Nuestro sistema computacional tendrá **2 memorias generales**: una interna (64 bloques de 32 bits) y un set de registros generales (32 registros de 32bits).
5. Características de la memorias:
    1. El set de **16 registros de 32bits** generales son para mantener los funcionamientos básicos de nuestro procesador.
    2. Se tendrán también 7 flags generales para el control de las operaciones de nuestro computador. Las 4 flags clásicas de la ALU y 3 flags de control de memorias para la seguridad.
6. Utilizaremos un sistema de configuración total, con un Excel, de forma que, el control total de la ejecución de nuestro programa dependerá totalmente del estado de la información de un Excel. Y nuestro programa, solo estará, mostrando la información adecuada SEGÚN esta este presente en el Excel.

Instrucciones De Seguridad:

1. Se tendrán las siguientes 4 memorias para el funcionamiento de seguridad de nuestro sistema:
    1. Memoria Protegida por Flag ‘L’:
        1. Un set de **8 bloques de 32 bits** para el manejo de nuestra contraseña de accesibilidad. (El flag de ‘L’ protege estos registros para su visualización y los flags de ‘S1’ y ‘S2’ proteger su escritura).
    2. Memoria Protegida por Flag ‘S1’ y ‘S2’:
        1. Un set de **10 registros de 32bits** para el proceso de cifrado/descifrado de nuestro programa. El Delta se guardará en uno de los registros del proceso de cifrado y el DELTA será: **0x9e3779b9.**
        2. Un banco de llaves criptográficas compuesto de **16 bloques de 32bits** (4 bloques por llave). Cada llave de encriptación será de **128bits.**
        3. Una **memoria dinámica**, en esta memoria se guardará la información aleatoria la cual se encriptará o desencriptara con TEA.
2. Se va a tener un par de flags extras los cuales son, dos “security flag”. Son dos para que una condición de funcionamiento de hardware, o sea que ambos flags tienen que estar coincidiendo en todo momento, SI UNO NO COINDICE, entonces se considera que esto es un error de hardware y no se aplican las condiciones de funcionamiento.
3. Se tendrá un tercer flag: "login flag", que es para asegurar la reservación del funcionamiento de "login“.
4. Se aplicará una contraseña, la cual, nos permitirá aplicar una acción de login y esta acción va a comparar si la contraseña es igual a la contraseña guardada. Esto permitirá la habilitación de los “security flag”.

Resumen de las memorias del programa:

1. Memoria General:
    1. 64 bloques de 32bits para propósito Multiple
2. Registros Generales:
    1. 16 registros de 32 bits para propósitos varios:
        1. R0 -> Registro de completos 0 estático INCAMBIABLE
        2. R1 -> Multiple propósito
        3. R2 -> Multiple propósito
        4. R3 -> Multiple propósito
        5. R4 -> Multiple propósito
        6. R5 -> Multiple propósito
        7. R6 -> Multiple propósito
        8. R7 -> Multiple propósito
        9. R8 -> Multiple propósito
        10. R9 -> Multiple propósito
        11. R10 -> Multiple propósito
        12. R11 -> Multiple propósito
        13. R12 -> Multiple propósito
        14. R13 -> Multiple propósito
        15. R14 -> Multiple propósito
        16. R15 -> Multiple propósito

1. Memoria de Login:
    1. 8 bloques de 32bits para guardar nuestra contraseña de 256bits
2. Registros de Seguridad:
    1. 9 registros de 32 bits para propósito múltiples de descifrado o cifrado:
        1. w1 -> Puntero de la memoria dinámica actual
        2. w2 -> Tamaño en bytes de la memoria dinámica con un LSR de #3: (x1, #3)
        3. w3 -> Puntero de memoria inicial (Controla el movimiento de los bloques)
        4. w4 -> Primer bloque de 32bits de contenido (v0)
        5. w5 -> Segundo bloque de 32bits de contenido (v1)
        6. w6 -> Registro general de suma
        7. w7 -> Registro de contador de rondas
        8. w8 -> Registro intermedio de operación
        9. w9 -> Registro intermedio de operación
        10. d0 -> Delta estático en un registro... (0x9E3779B9)
3. Memoria de la Bodega de Llaves Criptográficas:
    1. 16 bloques de 32bits para guardar nuestras llaves como memoria (4 bloques por cada llave):
        1. k0.0 -> llave 0 bloque 0 de 32bits
        2. k0.1 -> llave 0 bloque 1 de 32bits
        3. k0.2 -> llave 0 bloque 2 de 32bits
        4. k0.3 -> llave 0 bloque 3 de 32bits
        5. k1.0 -> llave 1 bloque 0 de 32bits
        6. k1.1 -> llave 1 bloque 1 de 32bits
        7. Etc.
4. Memoria Dinámica Segura:
    1. Va a poseer un tamaño desconocido por el hardware de nuestro programa, entonces aprovecharemos el High Level Programming Language para trabajar con esta.

Flags del Procesador:

Nombre	Descripción
N (Negative)	Se activa cuando el resultado es negativo (bit más significativo = 1)
Z (Zero)	Se activa cuando el resultado es cero
C (Carry)	Se activa cuando hay un acarreo en operaciones aritméticas
V (Overflow)	Se activa cuando hay un desbordamiento en operaciones con signo
S1 (Safe 1)	Se activa cuando se produce un login efectivo. Se apaga si se aplica alguna operación general.
S1 (Safe 2)	Se activa cuando se produce un login efectivo. Se apaga si se aplica alguna operación general.
L (Login)	Se activa solo, durante el intento de operación de Login.

Operaciones del procesador básicas

Operaciones Aritméticas:

Nombre	Descripción	Operación
ADD Rd, Rn, Rm	Suma entre registros	Rd ← Rn + Rm
ADDS Rd, Rn, k?.?	Suma entre registro y bloque de la 'Vault'	Rd ← Rn + k?.?
ADDI Rd, Rn, #?	Suma con inmediato	Rd ← Rn + Im
SUB Rd, Rn, Rm	Resta entre registros	Rd ← Rn - Rm
SUBI Rd, Rn, #?	Resta con inmediato	Rd ← Rn - Im
ADC Rd, Rn, Rm	Suma con acarreo entre registros	Rd ← Rn + Rm + C
ADCI Rd, Rn, #?	Suma con acarreo con inmediato	Rd ← Rn + Im + C
SBC Rd, Rn, Rm	Resta con acarreo entre registros	Rd ← Rn - Rm - !C
SBCI Rd, Rn, #?	Resta con acarreo con inmediato	Rd ← Rn - Im - !C
MUL Rd, Rn, Rm	Multiplicación entre registros	Rd ← Rn × Rm
MULI Rd, Rn, #?	Multiplicación con inmediato	Rd ← Rn × Im
DIV Rd, Rn, Rm	División entre registros	Rd ← Rn ÷ Rm
DIVI Rd, Rn, #?	División con inmediato	Rd ← Rn ÷ Im

13 Operaciones

Operaciones Lógicas:

Nombre	Descripción	Operación
AND Rd, Rn, Rm	AND Bit a bit entre registros	Rd ← Rn & Rm
ANDI Rd, Rn, #?	AND Bit a bit con inmediato	Rd ← Rn & Im
ORR Rd, Rn, Rm	OR Bit a bit entre registros	Rd ← Rn | Rm
ORRI Rd, Rn, #?	OR Bit a bit con inmediato	Rd ← Rn | Im
EOR Rd, Rn, Rm	XOR Bit a bit entre registros	Rd ← Rn ^ Rm
EORI Rd, Rn, #?	XOR Bit a bit con inmediato	Rd ← Rn ^ Im
BIC Rd, Rn, Rm	Bit Clear (AND con NOT) entre registros	Rd ← Rn & ~Rm
BICI Rd, Rn, #?	Bit Clear (AND con NOT) con inmediato	Rd ← Rn & ~Im
MVN Rd, Rm	Mover NOT entre registros	Rd ← ~Rm
MVNI Rd, #?	Mover NOT con inmediato	Rd ← ~Im

10 Operaciones

Desplazamientos y Rotaciones:

Nombre	Descripción	Operación
LSL Rd, Rn, Rm	Desplazamiento lógico a la izquierda con val de registro	Rd ← Rn << Rm
LSLI Rd, Rn, #?	Desplazamiento lógico a la izquierda con val de inmediato	Rd ← Rn << Im
LSR Rd, Rn, Rm	Desplazamiento lógico a la derecha con val de registro	Rd ← Rn >> Rm
LSRI Rd, Rn, #?	Desplazamiento lógico a la derecha con val de inmediato	Rd ← Rn >> Im
ASR Rd, Rn, Rm	Desplazamiento aritmético a la derecha con val de registro	Rd ← Rn >> Rm
ASRI Rd, Rn, #?	Desplazamiento aritmético a la derecha con val de inmediato	Rd ← Rn >> Im
ROR Rd, Rn, Rm	Rotación a la derecha con val de registro	Rd ← ROR(Rn, Rm)
RORI Rd, Rn, #?	Rotación a la derecha con val de inmediato	Rd ← ROR(Rn, Src2)

8 Operaciones

Operaciones de Movimiento de Datos*:*

Nombre	Descripción	Operación
MOV Rd, Rm	Mover con registros	Rd ← Rm
MOVI Rd, #?	Mover con inmediato	Rd ← Im
LDR Rd, G[Rn, #offset]	Cargar de memoria general	Rd ← MemG[Rn + offset]
LDR Rd, D[Rn, #offset]	Cargar de memoria dinámica	Rd ← MemD[Rn + offset]
STR Rd, G[Rn, #offset]	Guardar en memoria	MemG[Rn + offset] ← Rd
STR Rd, D[Rn, #offset]	Guardar en memoria	MemD[Rn + offset] ← Rd
LDRB Rd, G[Rn, #offset]	Cargar byte de memoria	Rd ← MemG[Rn + offset] (byte)
LDRB Rd, D[Rn, #offset]	Cargar byte de memoria	Rd ← MemD[Rn + offset] (byte)
STRB Rd, G[Rn, #offset]	Guardar byte en memoria	MemG[Rn + offset] ← Rd (byte)
STRB Rd, D[Rn, #offset]	Guardar byte en memoria	MemD[Rn + offset] ← Rd (byte)

10 Operaciones

Operaciones de Comparación:

Nombre	Descripción	Operación
CMP Rn, Rm	Comparación entre registros	Actualiza flags según (Rn - Rm)
CMPI Rn, #?	Comparación entre registro e inmediato	Actualiza flags según (Rn - Im)
CMN Rn, Rm	Comparar negativo entre registros	Actualiza flags según (Rn + Rm)
CMNI Rn, #?	Comparar negativo entre registro e inmediato	Actualiza flags según (Rn + Im)
TST Rn, Rm	Test de bits entre registros	Actualiza flags según (Rn & Rm)
TSTI Rn, #?	Test de bits entre registro e inmediato	Actualiza flags según (Rn & Im)
TEQ Rn, Rm	Test igualdad entre registros	Actualiza flags según (Rn ^ Rm)
TEQI Rn, #?	Test igualdad entre registro e inmediato	Actualiza flags según (Rn ^ Im)

8 Operaciones

Operaciones de Bifurcación:

Nombre	Descripción	Operación
B etiqueta	Salto incondicional	PC ← etiqueta
BEQ etiqueta	Salto si igual (Z=1)	Si Z=1, entonces PC ← etiqueta
BNE etiqueta	Salto si no igual (Z=0)	Si Z=0, entonces PC ← etiqueta
BLT etiqueta	Salto si menor que (N!=V)	Si N!=V, entonces PC ← etiqueta
BGT etiqueta	Salto si mayor que (Z=0 AND N=V)	Si Z=0 y N=V, entonces PC ← etiqueta

5 Operaciones

Operaciones Especiales*:*

Nombre	Descripción	Operación
SWI	Interrupción de software	Generar interrupción software
NOP	No operación	PC ← PC + 4
PRINTI G[Rn, #offset]	Imprimir entero desde memoria general	Console ← int(MemG[Rn + offset])
PRINTI V[Rn, #offset]	Imprimir entero desde memoria de la ‘vault’	Console ← int(MemV[Rn + offset])
PRINTI D[Rn, #offset]	Imprimir entero desde memoria dinámica	Console ← int(MemD[Rn + offset])
PRINTI P[Rn, #offset]	Imprimir entero desde memoria de password	Console ← int(MemP[Rn + offset])
PRINTR Rd	Imprimir entero desde registro	Console ← int(Rd)
PRINTS G[Rn, #limit]	Imprimir ASCII desde memoria (rango) general	Console ← ASCII(MemG[Rn] hasta MemG[#limit])
PRINTS V[Rn, #limit]	Imprimir ASCII desde memoria (rango) de la ‘vault’	Console ← ASCII(MemV[Rn] hasta MemV[#limit])
PRINTS D[Rn, #limit]	Imprimir ASCII desde memoria (rango) dinámica	Console ← ASCII(MemD[Rn] hasta MemD[#limit])
PRINTS P[Rn, #limit]	Imprimir ASCII desde memoria (rango) de password	Console ← ASCII(MemP[Rn] hasta MemP[#limit])
PRINTL "texto"	Imprimir literal ASCII	Console ← "texto"

12 Operaciones

Operaciones del procesador avanzadas

Operaciones Login*:*

Nombre	Descripción	Operación
AUTHCMP	Comparar para login con un val de memoria de login, R1-8 == P[#0 - #12] 1. Si el login es exitoso, se tendrá un timer de 10min en donde se habilitaran los flags de S1 y S2. (Debería de funcionar incluso en un “lights out”) 2. Si se falla el login, se sumara a un contador interno el fallo, si en un periodo de 5min se fallo 5 veces seguidas, entonces se bloquea el login por 24 horas.	Actualiza flags según (R1-8 == P[#0 - #12])
LOGOUT	Comando para intentar hacer un logout del sistema de seguridad del computador. Al activar este comando, el timer de login se reinicia a 0, y los flags de S1 y S2 regresan a 0 inmediatamente.	

2 Operaciones

Operaciones En Memoria Dinámica:

Nombre	Descripción	Operación
(Botón de cargado de nuestro programa)	Guardar en memoria y cerrarlo a un múltiplo de 64bits con 0's al final	"D[]"" -> Puntero de memoria inicial dinámica w2 -> Tamaño en bytes de la memoria dinámica con un LSR de #3: (x1, #3) //x1 sería el tamaño de la memoria (Esta operación debería alterar los flags…)

Operaciones De Guardado De Contraseña y Banco De Llaves:

Nombre	Descripción	Operación
STRK <clave>.<word> #valHex STRK 0.0 #0x12345678	Guarda 32bits de una llave, en una de sus 4 secciones (palabras)	k0.0 'Valt[#0 + #0]' = 0x12345678
STRK <clave>.<word> #valHex STRK 2.2 #0x12345678	Guarda 32bits de una llave, en una de sus 4 secciones (palabras)	k2.2 'Valt[#32 + #8]' = 0x12345678
STRK <clave>.<word> #valHex STRK 3.1 #0x12345678	Guarda 32bits de una llave, en una de sus 4 secciones (palabras)	k3.1 'Valt[#48 + #4]' = 0x12345678

Nombre	Descripción	Operación
STRPASS <word> #valHex STRPASS 0 #0x12345678	Guarda 32bits de una contraseña, en una de sus 8 secciones (palabras)	Password[#0] = 0x12345678
STRPASS <word> #valHex STRPASS 1 #0x12345678	Guarda 32bits de una contraseña, en una de sus 8 secciones (palabras)	Password[#4] = 0x12345678

24 Operaciones

Operaciones De Cifrado:

Nombre	Descripción	Operación
TEA #0, #valHex	Variables de uso para cada ronda	MOVI w6, #0xvalHex MOVI w7, #32 //Contador de rondas
TEAENC #1, k0/k1/k2/k3	Realiza la suma del delta y parte del primer termino v1	ADD w6, w6, d0 //Suma del delta a sum LSLI w8, w5, #4 //(v1 << 4) ADDS w8, w8, k?.0 //result + k0
TEAENC #2	Calculo (v1 + sum) y le hace XOR con el acumulado en w8.	ADD w9, w5, w6 //Suma de v1 + sum EOR w8, w8, w9 //XOR entre resultados
TEAENC #3, k0/k1/k2/k3	Calcula (v1 >> 5) + k1, luego XOR con lo ya presente en w8.	LSRI w9, w5, #5 //Rotación de 5 a v1 ADDS w9, w9, k?.1//Suma de resultado con k1 EOR w8, w8, w9 //XOR para tener resultado en w8
TEAENC #4, k0/k1/k2/k3	Calcula el nuevo valor de v0 e inicial los del v1	ADD w4, w4, w8 //Actualización de v0 LSLI w8, w4, #4 // (v0 << 4) ADDS w8, w8, k?.2 //result + k2
TEAENC #5	Calcula el nuevo valor de v0 e inicial los del v1	ADD w9, w4, w6 //(v0 + sum) EOR w8, w8, w9 //Xor de result
TEAENC #6, k0/k1/k2/k3	Calculo del XOR con (v0 >> 5) + k3	LSRI w9, w4, #5 //Rotación de 5 a v0 ADDS w9, w9, k?.3 //Suma de resultado con k3 EOR w8, w8, w9 //XOR del resultado final
TEAENC #7	Calculo final de v1 y revisión del bucle	ADD w5, w3, w8 //Resultado parcial nuevo de v1 SUBI w7, w7, #1 //Decremento de ronda

20 Operaciones

Operaciones De Descifrado:

Nombre	Descripción	Operación
TEAD #1, k0/k1/k2/k3	Realiza las 3 primeras operaciones del proceso de desencriptación:	LSLI w8, w4, #4 /* (v0 << 4) */ ADDS w8, w8, k?.2 /* + k2 */ ADD w9, w4, w6 /* v0 + sum */
TEAD #2, k0/k1/k2/k3	Realiza las 3 siguientes operaciones del proceso de desencriptación:	EOR w8, w8, w9 LSRI w9, w4, #5 /* (v0 >> 5) */ ADDS w9, w9, k?.3 /* + k3 */
TEAD #3	Realiza las 3 siguientes operaciones del proceso de desencriptación:	EOR w8, w8, w9 SUB w3, w3, w8 /* v1 -= temp */
TEAD #4, k0/k1/k2/k3	Realiza las 3 siguientes operaciones del proceso de desencriptación:	LSLI w8, w5, #4 /* (v1 << 4) */ ADDS w8, w8, k?.0 /* + k0 */ ADD w9, w3, w6 /* v1 + sum */
TEAD #5, k0/k1/k2/k3	Realiza las 3 siguientes operaciones del proceso de desencriptación:	EOR w8, w8, w9 LSRI w9, w5, #5 /* (v1 >> 5) */ ADDS w9, w9, k?.1 /* + k1 */
TEAD #6	Realiza las 3 siguientes operaciones del proceso de desencriptación:	EOR w8, w8, w9 SUB w4, w4, w8 /* v0 -= temp */
TEAD #7	Realiza las ultimas operaciones del proceso de desencriptación:	SUB w6, w6, d0 /* sum -= Δ */ SUBI w7, w7, #1

19 Operaciones

Ejemplo de encriptación y desencriptación:

TEA Cifrado de memoria aleatoria

```
STRS(Externo)

tea_decrypt_buf:
        MOV w3, R0  //desplazamiento del puntero de memoria inicial
.Lblock_Loop:  //inicio bucle por bloque
        LDR w4, D[w3] //Carga del bloque v0
        LDR w5, D[w3, #4] //Carga del bloque v1
        TEA   #0, #0x0 //Suma inicial en 0
.Lloop:             ; — núcleo de 32 rondas TEA —
        TEAENC   #1, k0
        TEAENC   #2
        TEAENC   #3, k0
        TEAENC   #4, k0
        TEAENC   #5
        TEAENC   #6, k0
        TEAENC   #7
        BNE .Lloop //Revisión de Loop
        STR w4, D[w3]  //Guardar block 32bits
        STR w5, D[w3, #4] //Guardar segundo block 32bits
        ADD w3, w3, #8 //Incremente del siguiente bloque
        SUBS w2, w2, #1  //Restamos un bloque de 64bits
        BNE .Lblock_Loop //Saltamos si aun quedan bloques
.Ldone:             ; ===  fin del bucle  ===
        PRINTS D[R0, #?]          ; print de mem dinamic
```

TEA Descifrado de memoria aleatoria

```
STRS(Externo)

tea_decrypt_buf:
        MOV w3, R0  //desplazamiento del puntero de memoria inicial
.Lblock_Loop:  //inicio bucle por bloque
        LDR w4, D[w3] //Carga del bloque v0
        LDR w5, D[w3, #4] //Carga del bloque v1
        TEA   #0, #0xC6EF3720 //Suma inicial en delta*32
.Lloop:             ; — núcleo de 32 rondas TEA —
        TEAD   #1, k0
        TEAD   #2, k0
        TEAD   #3
        TEAD   #4, k0
        TEAD   #5, k0
        TEAD   #6
        TEAD   #7
        BNE .Lloop //Revision de Loop
        STR w4, D[w3]  //Guardar block 32bits
        STR w5, D[w3, #4] //Guardar segundo block 32bit
        ADD w3, w3, #8 //Incremente del siguiente bloque
        SUBS w2, w2, #1  //Restamos un bloque de 64bits
        BNE .Lblock_Loop
.Ldone:             ; ===  fin del bucle  ===
        PRINTS D[R0, #?]    ; print de mem dinamic
```

Codificación de las instrucciones

Ejemplo de división de bits de identificación en nuestro ISA

Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49 (Identificación del Rd)	Bit 48-43 (Identificación del Rn)	Bit 42-37 (Identificación del Rm)	Bit 36-0
ADD Rd, Rn, Rm	000000000	?00000 (El ? Es para identificar si es un R o un w, y el resto es para saber el numero) Seria: 1 para R, 0 para w.	?00000 (El ? Es para identificar si es un R o un w, y el resto es para saber el numero)	?00000 (El ? Es para identificar si es un R o un w, y el resto es para saber el numero)	000000…
SUB Rd, Rn, Rm	000000001	?00000	?00000	?00000	000000…
ADC Rd, Rn, Rm	000000010	?00000	?00000	?00000	000000…
SBC Rd, Rn, Rm	000000011	?00000	?00000	?00000	000000…
MUL Rd, Rn, Rm	000000100	?00000	?00000	?00000	000000…
DIV Rd, Rn, Rm	000000101	?00000	?00000	?00000	000000…
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49 (Identificación del Rd)	Bit 48-43 (Identificación del Rn)	Bit 42-11 (Identificación del Im)	Bit 11-0
ADDI Rd, Rn, #?	000000110	?00000	?00000	00000000000000000000000000000000	000000…
SUBI Rd, Rn, #?	000000111	?00000	?00000	00000000000000000000000000000000	000000…
ADCI Rd, Rn, #?	000001000	?00000	?00000	00000000000000000000000000000000	000000…
SBCI Rd, Rn, #?	000001001	?00000	?00000	00000000000000000000000000000000	000000…
MULI Rd, Rn, #?	000001010	?00000	?00000	00000000000000000000000000000000	000000…
DIVI Rd, Rn, #?	000001011	?00000	?00000	00000000000000000000000000000000	000000…

Ejemplo de división de bits de identificación en nuestro ISA					
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49 (Identificación del Rd)	Bit 48-43 (Identificación del Rn)	Bit 42-37 (Identificación del Rm)	Bit 36-0
AND Rd, Rn, Rm	000001100	?00000	?00000	?00000	000000…
ORR Rd, Rn, Rm	000001101	?00000	?00000	?00000	000000…
EOR Rd, Rn, Rm	000001110	?00000	?00000	?00000	000000…
BIC Rd, Rn, Rm	000001111	?00000	?00000	?00000	000000…
MVN Rd, Rm	000010000	?00000	?00000	?00000	000000…
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49 (Identificación del Rd)	Bit 48-43 (Identificación del Rn)	Bit 42-11 (Identificación del Im)	Bit 11-0
ANDI Rd, Rn, #?	000010001	?00000	?00000	00000000000000000000000000000000	000000…
ORRI Rd, Rn, #?	000010010	?00000	?00000	00000000000000000000000000000000	000000…
EORI Rd, Rn, #?	000010011	?00000	?00000	00000000000000000000000000000000	000000…
BICI Rd, Rn, #?	000010100	?00000	?00000	00000000000000000000000000000000	000000…
MVNI Rd, #?	000010001	?00000	?00000	00000000000000000000000000000000	000000…

Ejemplo de división de bits de identificación en nuestro ISA					
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49 (Identificación del Rd)	Bit 48-43 (Identificación del Rn)	Bit 42-37 (Identificación del Rm)	Bit 36-0
LSL Rd, Rn, Rm	000010101	?00000	?00000	?00000	000000…
LSR Rd, Rn, Rm	000010110	?00000	?00000	?00000	000000…
ASR Rd, Rn, Rm	000010111	?00000	?00000	?00000	000000…
ROR Rd, Rn, Rm	000011000	?00000	?00000	?00000	000000…
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49 (Identificación del Rd)	Bit 48-43 (Identificación del Rn)	Bit 42-11 (Identificación del Im)	Bit 11-0
LSLI Rd, Rn, #?	000011001	?00000	?00000	00000000000000000000000000000000	000000…
LSRI Rd, Rn, #?	000011010	?00000	?00000	00000000000000000000000000000000	000000…
ASRI Rd, Rn, #?	000011011	?00000	?00000	00000000000000000000000000000000	000000…
RORI Rd, Rn, #?	000011100	?00000	?00000	00000000000000000000000000000000	000000…

Ejemplo de división de bits de identificación en nuestro ISA

Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49 (Identificación del Rd)	Bit 48-43 (Identificación del Rn)	Bit 42-37 (Identificación del Rm)	Bit 36-0
MOV Rd, Rm	000011101	?00000	?00000	?00000	0000…
Operación actual	**Bit 63-55 (Identificación de Op)**	**Bit 54-49 (Identificación del Rd)**	**Bit 48-43 (Identificación del Rn)**	**Bit 42-11 (Identificación del Im)**	**Bit 11-0**
MOVI Rd, #?	000011110	?00000	?00000	00000000000000000000000000000000	0000…

Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49 (Identificación del Rd)	Bit 48 (De cual Mem)	Bit 47-42 (Identificación del Rn)	Bit 41-10 (Val #offset)	Bit 9-0
LDR Rd, G[Rn, #offset]	000011111	?00000	0 (General)	?00000	00000000000000000000000000000000	0000…
LDR Rd, D[Rn, #offset]	000100000	?00000	1 (Dinámica)	?00000	00000000000000000000000000000000	0000…
STR Rd, G[Rn, #offset]	000100001	?00000	0	?00000	00000000000000000000000000000000	0000…
STR Rd, D[Rn, #offset]	000100010	?00000	1	?00000	00000000000000000000000000000000	0000…
LDRB Rd, G[Rn, #offset]	000100011	?00000	0	?00000	00000000000000000000000000000000	0000…
LDRB Rd, D[Rn, #offset]	000100100	?00000	1	?00000	00000000000000000000000000000000	0000…
STRB Rd, G[Rn, #offset]	000100101	?00000	0	?00000	00000000000000000000000000000000	0000…
STRB Rd, D[Rn, #offset]	000100110	?00000	1	?00000	00000000000000000000000000000000	0000…

Ejemplo de división de bits de identificación en nuestro ISA				
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49 (Identificación del Rn)	Bit 48-43 (Identificación del Rm)	Bit 42-0
CMP Rn, Rm	000100111	?00000	?00000	000000…
CMN Rn, Rm	000101000	?00000	?00000	000000…
TST Rn, Rm	000101001	?00000	?00000	000000…
TEQ Rn, Rm	000101010	?00000	?00000	000000…
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49 (Identificación del Rn)	Bit 48-17 (Identificación del Im)	Bit 16-0
CMPI Rn, #?	000101011	?00000	00000000000000000000000000000000	000000…
CMNI Rn, #?	000101100	?00000	00000000000000000000000000000000	000000…
TSTI Rn, #?	000101101	?00000	00000000000000000000000000000000	000000…
TEQI Rn, #?	000101110	?00000	00000000000000000000000000000000	000000…

Ejemplo de división de bits de identificación en nuestro ISA				
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54 (Identificación del valor positivo o negativo)	Bit 53-22 (Identificación del Im que se suma o resta a PC)	Bit 21-0
B etiqueta	000101111	?	00000000000000000000000000000000	?00000
BEQ etiqueta	000110000	?	00000000000000000000000000000000	?00000
BNE etiqueta	000110001	?	00000000000000000000000000000000	?00000
BLT etiqueta	000110010	?	00000000000000000000000000000000	?00000
BGT etiqueta	000110011	?	00000000000000000000000000000000	?00000

Ejemplo de división de bits de identificación en nuestro ISA					
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-0			
SWI	000110100	00000…			
NOP	000110101	00000…			
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-53 (De cual Mem)	Bit 52-47 (Identificación del Rn)	Bit 46-15 (Valor del #offset)	Bit 14-0
PRINTI G[Rn, #offset]	000110110	00 (General)	?00000	00000000000000000000000000000000	00000…
PRINTI V[Rn, #offset]	000110111	01 (‘Vault’)	?00000	00000000000000000000000000000000	00000…
PRINTI D[Rn, #offset]	000111000	10 (Dinámica)	?00000	00000000000000000000000000000000	00000…
PRINTI P[Rn, #offset]	000111001	11 (Password)	?00000	00000000000000000000000000000000	00000…
PRINTS G[Rn, #limit]	000111010	00	?00000	00000000000000000000000000000000	00000…
PRINTS V[Rn, #limit]	000111011	01	?00000	00000000000000000000000000000000	00000…
PRINTS D[Rn, #limit]	000111100	10	?00000	00000000000000000000000000000000	00000…
PRINTS P[Rn, #limit]	000111101	11	?00000	00000000000000000000000000000000	00000…

Ejemplo de división de bits de identificación en nuestro ISA			
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49	Bit 48-0
PRINTR Rd	000111110	?00000	00000…
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-0 (Valor de ASCII en bin)	
PRINTL "texto"	000111111	00000…	

Ejemplo de división de bits de identificación en nuestro ISA		
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-0
AUTHCMP	001000000	00000…
LOGOUT	001000001	00000…

Ejemplo de división de bits de identificación en nuestro ISA					
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-53 (Identifica cual llave es k0,k1,k2,k3)	Bit 52-51 (Identifica cual bloque de los 4 posibles es k0,k1,k2,K3)	Bit 50-19 (Identifica el valor de valHex)	Bit 18-0
STRK <clave>.<word> #valHex	001000010	??	??	00000000000000000000000000000000	000…
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-52 (Identifica cual bloque de contraseña)	Bit 51-20 (Identifica el valor de valHex)		Bit 19-0
STRPASS <word> #valHex	001000011	???	00000000000000000000000000000000		000…

Ejemplo de división de bits de identificación en nuestro ISA (ESTAS SON OPERACIONES DE NUESTRO COMPILADOR, O SEA, LA TRADUCCION DE ESTO A BINARIO SON LAS INSTRUCCIONES QUE SE IMPLEMENTARÁN, LAS VARIAS)			
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-23 (Identifica el valor de del valHex)	Bit 22-0
TEA #0, #valHex	001000100	00000000000000000000000000000000	0000…
Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-53 (Identifica cual llave se utiliza)	Bit 52-0
TEAENC #1, k0/k1/k2/k3	001000101	??	000…
TEAENC #3, k0/k1/k2/k3	001000110	??	000…
TEAENC #4, k0/k1/k2/k3	001000111	??	000…
TEAENC #6, k0/k1/k2/k3	001001000	??	000…

Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-0
TEAENC #2	001001001	000000…
TEAENC #5	001001010	000000…
TEAENC #7	001001011	000000…

Ejemplo de división de bits de identificación en nuestro ISA (ESTAS SON OPERACIONES DE NUESTRO COMPILADOR, O SEA, LA TRADUCCION DE ESTO A BINARIO SON LAS INSTRUCCIONES QUE SE IMPLEMENTARÁN LAS VARIAS)

Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-53 (Identifica cual llave se utiliza)	Bit 52-0
TEAD #1, k0/k1/k2/k3	001001100	??	000…
TEAD #2, k0/k1/k2/k3	001001101	??	000…
TEAD #4, k0/k1/k2/k3	001001110	??	000…
TEAD #5, k0/k1/k2/k3	001001111	??	000…

Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-0
TEAD #3	001010000	000000…
TEAD #6	001010001	000000…
TEAD #7	001010010	000000…

Operación actual	Bit 63-55 (Identificación de Op)	Bit 54-49 (Identificación del Rd)	Bit 48-43 (Identificación de Rn)	Bit 42-39 (Identificación del k?.?)	Bit 38-0
ADDS Rd, Rn, k?.?	001010011	?00000	?00000	???? (Los primeros ?? son para la llave, los otros para el bloque)	000000…

Diagramas del Funcionamiento Total

Desarrollo del Hardware Pipeline con Hazard:

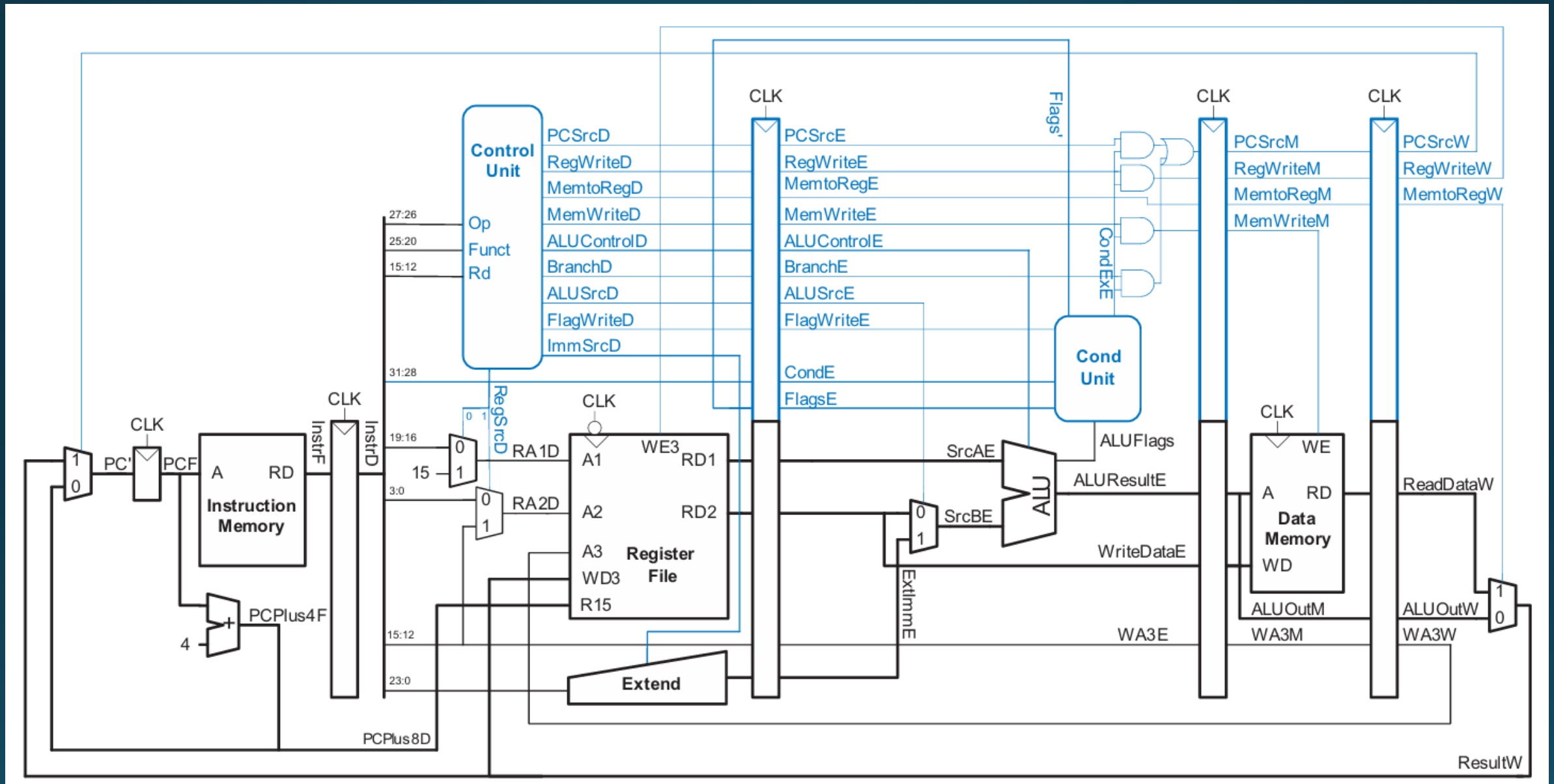

Figure 7.47 Pipelined processor with control

Siguiente Paso hacer un diagrama oficial, que incluya todos los componentes que nuestro procesador va a requerir (incluye los componentes extra de login)

Construir un Excel que lleve el control de los nombres y valores de TODO…

Hace falta hacer el compilador para transformar nuestra op a binario

Determinar cuantos bits se ocupan para las operaciones