Trabajo Práctico Nº 7
MICROARQUITECTURA

Camino de datos

1. Plantee el esquema de una estructura tipo bus en un sentido general y luego identifique los componentes de la microarquitectura ARC que conforman el "bus de address", el "bus de datos" y el "bus de control" en los buses A B y C. de la misma

2. El registro *%r0* puede ser diseñado por medio de buffers tri-state exclusivamente. Proponga el diagrama cicuital correspondiente.

3. Proponga un circuito que implemente el registro de instrucciones. Considere FF-D del tipo maestro-esclavo e indique todas las conexiones con el resto de los componentes.

4. Diseñar una unidad aritmético-lógica de 4 bits que tenga capacidad de realizar las operaciones addcc, andcc y orcc. Se piden dos soluciones alternativas:
   (a) circuitos dedicados (sumador y operadores lógicos)
   (b) lookup table
5. Plantear el esquema circuital de una microarquitectura ARC basada en 2 buses solamente.

Unidad de control

6. Defina ciclo de búsqueda-ejecución y explique la forma en que este es implementado en la microarquitectura ARC microprogramada. Indique los pasos clave del microprograma.

7. En un procesador ARC el registro Program Counter apunta a la siguiente instrucción guardada en RAM:

   ```
   addcc  %r10, 48, %r1
   ```

   Detallar los pasos de microprograma que la decodifican y para cada uno de ellos indicar los valores presentes en las entradas y en las salidas de cada uno de los siguientes bloques funcionales:
   - multiplexor de direcciones de la memoria de control
   - incrementador de direcciones de la memoria de control
   - decodificadores de los buses A B y C
   - multiplexores que intervienen en la decodificación de los buses A B y C
   - multiplexor de datos del bus C
   - bus de datos A B y C
   - entrada de direcciones del módulo de memoria RAM

   Considere que antes de la ejecución de esta instrucción %r10 = %r1 = 0

8. Idem ejercicio anterior considerando que la instrucción a decodificar es

   ```
   ld  %r10, %r1, %r7
   ```

   Considere que antes de la ejecución de esta instrucción: %r10=3000 , %r1=8, %r7=100

9. Especificar los problemas de funcionamiento que surgirían en el caso de que se conectara a tierra la entrada 'acknowledge' del incrementador de direcciones de la memoria de control

10. Especificar los problemas de funcionamiento que surgirían en caso de que el bit 13 del MIR quedara permanentemente conectado a la tensión de alimentación.

11. Proponer un diagrama circuital para el bloque "lógica de control de saltos" de la microarquitectura ARC

12. Proponer un diagrama circuital para el bloque "multiplexor de la memoria de control" de la microarquitectura ARC

Microcódigo

13. Indique el contenido binario en las posiciones 60 y 61 del MIR a fin de que comprendan las siguientes microinstrucciones:

    60: R[temp0] = □NOR(R[0],R[temp0]); IF Z THEN GOTO 64;

    61: R[rd] = □INC(R[rs1]);

    Complete con 0´s los campos que no sean necesarios e indique eventuales consecuencias de completarlos con otro valor.

14. Proponga un patrón de 0´s y 1´s en los campos C y CMUX del registro de microinstrucciones de manera que ningún registro de propósito general cambie su contenido.

15. Considerando que la siguiente tabla almacena el microcódigo de una unidad ARC microprogramada indicar las tres correspondientes microinstrucciones en forma de mnemonicos

A	AMUX	B	BMUX	C	CMUXDR	ALU	COND	JUMP ADDR
10010	1	0000000	0	10000	10001	000	11000	00000000000
0000001	1	10000	1	01000	01000	001	10001	11100000001
0000001	0	0000000	1	10000	10001	000	10001	11100010010

16. Un microprograma requiere de un salto condicional basado en el bit 13 del registro de instrucciones. Se proponen dos soluciones, los respectivos segmentos de microcódigo se presentan a continuación:

    - 1792: R[temp0] = ADD(R[rs1],R[rs2]);
      IF R[IR[13]] THEN GOTO 1794;

    - 1792: IF R[IR[13]] THEN GOTO 1794;
      R[temp0] = ADD(R[rs1],R[rs2]);

    a) Indique eventuales ventajas o inconvenientes de una solución u otra.
    b) De ejemplos de microprogramación en que el valor del bit 13 condiciona el flujo de microprograma

17. El siguiente microcódigo corresponde a la instrucción *call* del Assembler ARC.

    1280: R[15] = AND(R[pc],R[pc]);
    1281: R[temp0] = ADD(R[ir],R[ir]);
    1282: R[temp0] = ADD(R[temp0],R[temp0]);
    1283: R[pc] = ADD(R[pc],R[temp0]);
    GOTO 0

    Reescribirlo de modo que se usen sólo 3 líneas de microcódigo en vez de 4. Utilizar la operación LSHIF2.

18. Dado el siguiente microcódigo correspondiente a la instrucción *subcc* del Assembly ARC:

    1584: R[temp0] = SEXT13(R[ir]);
    IF IR[13] THEN GOTO 1586;
    1585: R[temp0] = OR(R[0],R[rs2]);
    1586: R[temp0] = NOR(R[temp0], R[0]);
    1587: R[temp0] = INC(R[temp0]);
    1588: R[rd] = □ADDCC(R[rs1],R[temp0]);
    GOTO 2047;

    Calcule cuántos ciclos de reloj demora el procesador en ejecutar
    subcc %r1,%r4,%r6

19. Indique el contenido binario del Control Store entre las direcciones 1584 y 1588 de acuerdo al microcódigo del ejercicio anterior

20. El standard SPARC incluye la instrucción *addxcc (add with carry)* la cual es similar a *addcc* salvo que además de los sumandos estándar suma también el contenido del bit de carry. Se pide:

    a) Dar un ejemplo de su aplicación en el contexto de un programa que realiza la suma de números de 64 bits.

    b) Detalle los cambios que deberían ser introducidos en la memoria de control de un procesador ARC para que *addxcc* forme parte de su set de instrucciones. Para ello considerar que esta instrucción debe cumplir con el formato de operaciones aritmético-lógicas con un op3 igual a *010011*.

21. Según se propone en el libro de Murdocca-Heuring, el microcódigo guardado en la dirección 2047 de la memoria de control es el siguiente:

    2047: R[pc] = INCPC(R[pc]); GOTO 0;

    Proponga un código alternativo que no incluya GOTO 0 y que cumpla la misma función.

22. El siguiente microcódigo decodifica instrucciones de salto condicional. El mismo incluye intencionalmente un error en la posición 12 en el modo con que el PC es actualizado. Identifique el error y proponga una solución.

```
      / Branch instructions: ba, be, bcs, bvs, bneg
1088: GOTO 2;                              / Decoding tree for branches
   2: R[temp0] ← LSHIFT10(R[ir]);          / Sign extend the 22 LSB's of %temp0
   3: R[temp0] ← RSHIFT5(R[temp0]);        / by shifting left 10 bits, then right 10
   4: R[temp0] ← RSHIFT5(R[temp0]);        / bits. RSHIFT5 does sign extension.
   5: R[ir] ← RSHIFT5(R[ir]);              / Move COND field to IR[13] by
   6: R[ir] ← RSHIFT5(R[ir]);              / applying RSHIFT5 three times. (The
   7: R[ir] ← RSHIFT5(R[ir]);              / sign extension is inconsequential.)
   8: IF R[IR[13]] THEN GOTO 12;           / Is it ba?
      R[ir] ← ADD(R[ir],R[ir]);
   9: IF R[IR[13]] THEN GOTO 13;           / Is it not be?
      R[ir] ← ADD(R[ir],R[ir]);
  10: IF Z THEN GOTO 12;                   / Execute be
      R[ir] ← ADD(R[ir],R[ir]);
  11: GOTO 2047;                           / Branch for be not taken
  12: R[pc] ← ADD(R[pc],R[temp0]);         / Branch is taken
      GOTO 0;
  13: IF R[IR[13]] THEN GOTO 16;           / Is it bcs?
      R[ir] ← ADD(R[ir],R[ir]);
  14: IF C THEN GOTO 12;                   / Execute bcs
  15: GOTO 2047;                           / Branch for bcs not taken
  16: IF R[IR[13]] THEN GOTO 19;           / Is it bvs?
  17: IF N THEN GOTO 12;                   / Execute bneg
  18: GOTO 2047;                           / Branch for bneg  not taken
  19: IF V THEN GOTO 12;                   / Execute bvs
  20: GOTO 2047;                           / Branch for bvs not taken
2047: R[pc] ← INCPC(R[pc]); GOTO 0;        / Increment %pc and start over
```