P1

a)

i. ¿De cuantos kilobytes es el chip de memoria ROM?
R: Ya que ocupa los buses de direcciones A15 – A1 y los buses de datos de D15 – D0, podemos saber que el chip es de 2^15 * 2bytes = 2^16b = 2^6Kb = 64Kb

ii. ¿En qué rango de direcciones se ubica la ROM?
R: Ya que vemos un AND de las líneas A19 – A16 para el CS, la ROM está en el rango de direcciones [2^19 + 2^18 + 2^17 + 2^16, 2^20[, es decir, [960Kb , 1Mb[

iii. ¿De cuantos kilobytes es cada chip de memoria SRAM?
R: Ya que ocupan los buses de direcciones A17 – A1 y sus bus de datos son de 1byte, podemos saber que el chip es de 2^17 * byte = 2^17b = 2^7Kb = 128Kb

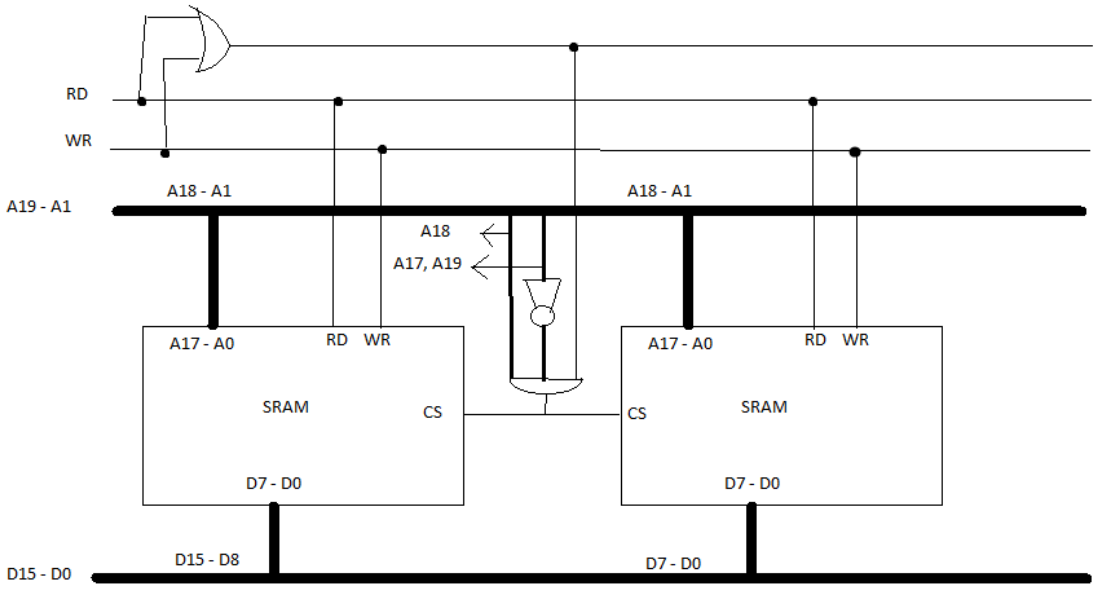
iv. ¿En qué rango de direcciones se ubica la SRAM?
R: Ya que vemos un AND de las líneas A19 – A18 negadas para el CS y las SRAM están en paralelo, se ubican en el rango de direcciones [0, 2^18[, es decir, [0Kb , 256Kb[

v. ¿Cuánta es la máxima cantidad de memoria, en kilobytes, que puede direccionar el procesador?
R: Puede direccionar 2^19 palabras de 2bytes, por lo que la cantidad total es 2^19 * 2b = 1Mb

vi. ¿Por qué el procesador no tiene la línea de dirección A0?
R: Por el alineamiento de memoria. Este ordenador es de 16 bits (2 bytes), por lo que la línea A0 estará reservada para direccionar el byte que se debe leer de la palabra.

vii. ¿Después de encenderse el procesador, qué puede decir acerca de la dirección de la primera instrucción que ejecuta este procesador?
R: Será la dirección 0, correspondiente al fetch. Esto se obtiene de la ROM, que debería contener la BIOS y/o el sistema operativo que usará el ordenador para ponerse en marcha.

b) En la figura se muestra los dos chips que se agregan. Ya que son de 256Kb, ocupan las líneas A18 – A1, pero sólo se activan si la línea A18 está activada y las A17 y A19 desactivadas, así el rango de direcciones es [256Kb, 640Kb[.



P2

a) Si la instrucción es LDRPC, Y-SEL tiene que entregar en su salida los bits desde el 0 hasta el 18 de lo que contiene IR, pero Y-SEL actualmente sólo posee cuatro instrucciones; por lo tanto, antes que todo, Y-SEL ahora debería recibir instrucciones de 3 bits en vez de 2 para que pueda tener más de 4 instrucciones; luego, una quinta instrucción (que podría ser con el código binario 100) tendría que leerla un multiplexor para que Y-SEL haga la instrucción: INST[18 – 0], que sería simplemente dejar pasar esas líneas y las demás no, así por su salida Y, estaría entregando el desplazamiento (el primer parámetro) que se debe sumar al registro PC. De esta manera, las instrucciones @0, @4, @INST y @DISP siguen estando disponibles y haciendo exactamente lo mismo que antes.
   A continuación se explica cómo se podrían implementar estas instrucciones luego de pasar por el multiplexor:
   - @0 (000): Se conecta un 0 a todas las líneas de la salida Y.
   - @4 (001): Se conecta un 0 directamente a todas las líneas de Y, excepto a Y2, que en vez de ir conectado directamente, iría negado.
   - @INST (010): la entrada REG y los 13 bits menos significativos de INST van conectados a un segundo multiplexor, que deja pasar a REG si INST13 es 1 y a los bits nombrados de INST si es 0.
   - @DISP (011): los 24 bits menos significativos de INST van conectados directamente a la salida Y.

- @LDRPC (100): los 19 bits menos significativos de INST van conectados directamente a la salida Y.

Hago esta última explicación de cada operación de OP-Y-SEL ya que un dibujo no sería tan claro respecto a las líneas que se conectan… y un circuito en logisim era mucho más trabajo

b) Luego de la implementación de LDRPC, las instrucciones de la unidad de control para este ciclo son:
   - **(SEL-REG =0, para que el multiplexor deje pasar a PC)**
   - OP-Y-SEL<-@LDRPC (para que Y-SEL entregue el desplazamiento, los primeros 19 bits que están en IR)
   - OP-ALU<-@ADD (para que sume PC con el desplazamiento
   - **(SEL-D=0, para que el multiplexor deje pasar PC + desplazamiento)**
   - WR-PC (para que PC escriba el resultado de la suma en ALU)