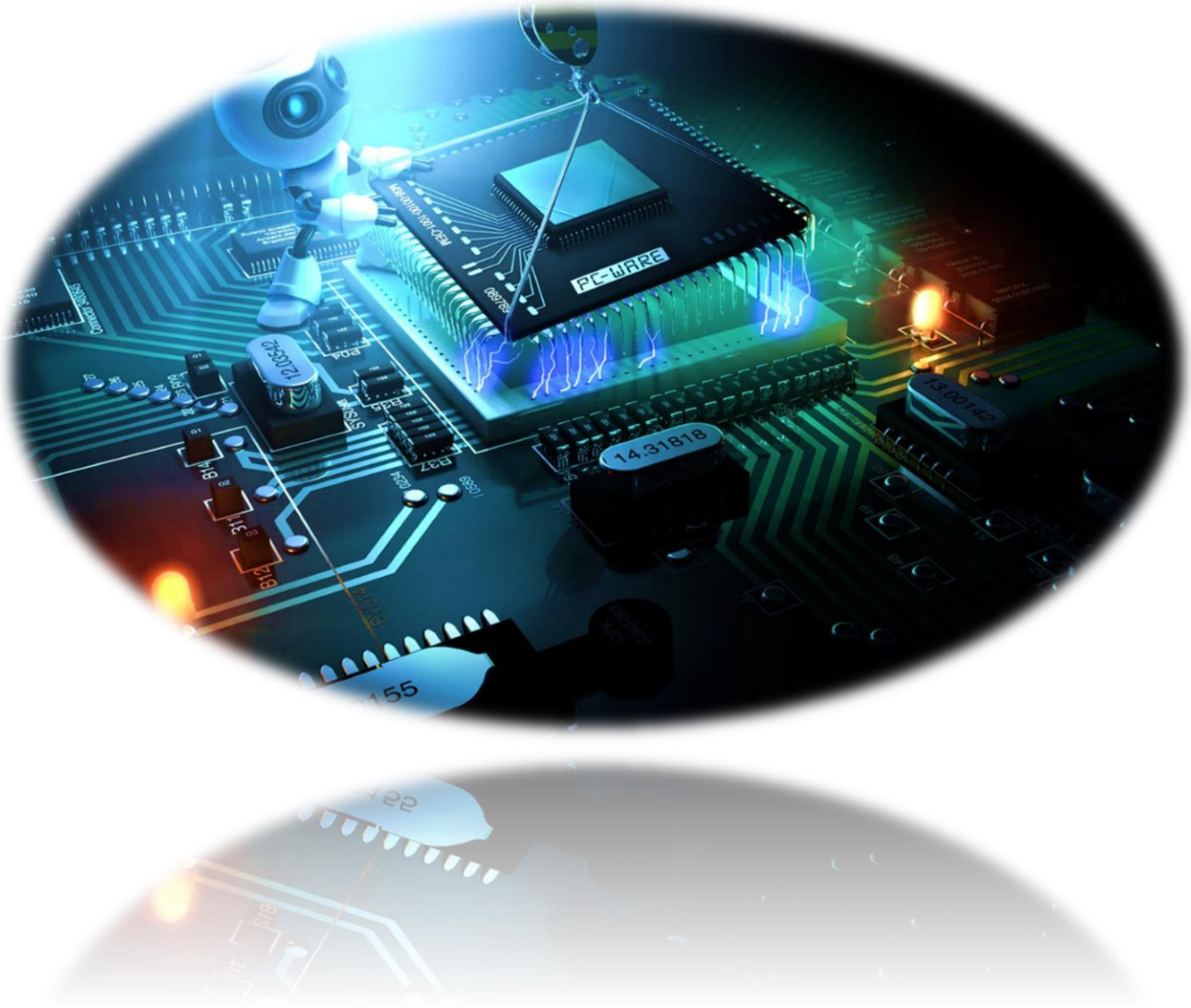
PRÁCTICA 3

ESTRUCTURA DE COMPUTADORES



BRYAN MORENO PICAMÁN

Contenido

Diario de trabajo

A continuación se detalla un diario de trabajo con los días dedicados a la práctica y las partes que se han desarrollado:

- 7 noviembre.- Primera lectura del tutorial de prácticas, y realización de pruebas que se indican, también primera lectura del guion de prácticas.
- 8 noviembre.- Segunda lectura del guion, primeras pruebas y comienzo de los códigos de la práctica en clase.
- 15 noviembre.- Realización de los códigos de la práctica, pruebas iniciales de comprobación de funcionamiento, preguntar de dudas en clases de prácticas.
- 18 noviembre.- Finalización de los códigos y terminar de comentar las parte que me faltaban
- 22 noviembre.- Realización de preguntas del cuestionario.
- 24 noviembre.- Realización de memoria de la práctica y entrega

Nota: Ordenador usado para las medidas es:

Intel(R) Core(TM) i5-4210H CPU @ 2.90GHz, 8Gb RAM, 1TB HDD.

POPCOUNT

Después de realizar las modificaciones a los códigos y obtener las salidas pertinentes acabamos con unas tablas de datos de la siguiente manera:

Optimización -O0	0	1	2	3	4	5	6	7	8	9	10	Media
popcount1 (en lenguaje C - for)	1166284	1167516	1201011	1168144	1160555	1168482	1161180	1163912	1160056	1166152	1166432	1168344
popcount2 (en lenguaje C - while)	1259383	1265737	1259812	1260095	1257052	1255685	1252790	1259917	1253794	1260887	1264934	1259070,3
popcount3 (leng.ASM - cuerpo while)	407593	404187	403117	406188	398969	398068	401717	399063	401065	398575	398789	400973,8
popcount4 (1.CS:APP 3.49 – group 8b)	311235	312537	314558	312730	312417	315014	313619	315611	313051	312096	313259	313489,2
popcount5 (asm SSE3 - pshfub 128b)	13084	13116	13074	12996	12970	13033	13015	13039	13166	13291	13106	13080,6
popcount5V2(Mejora acceso a memoria)	4251	3706	2827	3035	3093	3019	3192	3401	3440	3723	3984	3342
popcount6 (asm SSE4 - popcount 32b)	43863	45321	42942	42904	43000	42884	42867	42904	42920	42883	42933	43155,8
popcount7 (asm SSE4 - popcnt 2x32b)	31224	26739	26717	26660	26656	26610	26592	26607	26687	26640	26628	26653,6

Optimización -O1	0	1	2	3	4	5	6	7	8	9	10	Media
popcount1 (en lenguaje C - for)	239321	227875	238910	242140	234315	227405	224309	236087	238393	232326	232001	233376,1
popcount2 (en lenguaje C - while)	356981	352271	359768	353620	352449	352420	349733	350757	364118	354529	353814	354347,9
popcount3 (leng.ASM - cuerpo while)	323988	320138	324224	321421	322066	322972	317717	323043	322545	321271	319699	321509,6
popcount4 (1.CS:APP 3.49 – group 8b)	91273	86630	86816	89295	86063	87122	85988	90558	85872	86769	86988	87210,1
popcount5 (asm SSE3 - pshfub 128b)	6655	6645	6618	7134	7124	6997	7128	7868	6653	6683	6625	6947,5
popcount5V2(Mejora acceso a memoria)	191	191	200	223	240	222	265	202	227	240	192	220,2
popcount6 (asm SSE4 - popcount 32b)	7825	7488	7635	7791	7630	7645	7953	8464	7983	7751	7809	7814,9
popcount7 (asm SSE4 - popcnt 2x32b)	15223	15107	15125	15294	15091	15585	15393	15238	15164	15099	15118	15221,4

Optimización -O2	0	1	2	3	4	5	6	7	8	9	10	Media
popcount1 (en lenguaje C - for)	238580	236730	232746	241885	226517	245010	222504	232525	213245	228934	229925	231002,1
popcount2 (en lenguaje C - while)	291029	286970	287653	283314	288522	286095	284999	280825	282059	279634	275440	283551,1
popcount3 (leng.ASM - cuerpo while)	361600	360910	361716	352113	352647	357598	357406	354107	357718	352093	353848	356015,6
popcount4 (1.CS:APP 3.49 – group 8b)	90011	87950	92774	87186	88736	87708	88090	87764	89036	86890	88549	88468,3
popcount5 (asm SSE3 - pshfub 128b)	7161	7066	6911	7147	6756	7253	6640	6610	11348	6849	6818	7339,8
popcount5V2(Mejora acceso a memoria)	229	169	230	204	223	191	175	211	248	260	252	216,3
popcount6 (asm SSE4 - popcount 32b)	8227	7488	7503	7528	8023	7981	7842	7493	9462	7500	8108	7892,8
popcount7 (asm SSE4 - popcnt 2x32b)	15257	15083	15302	15143	15200	15393	15276	15275	15304	15189	15260	15242,5

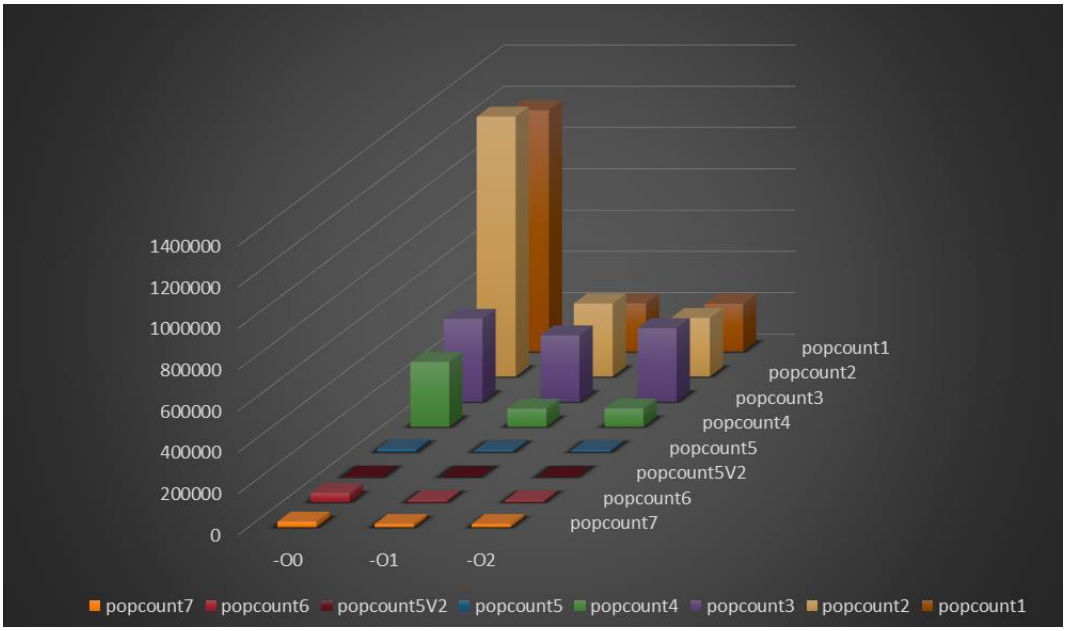
Como en la imagen no se aprecia bien se añadirán los archivos Excel para su examen posterior en caso de necesitarlo. Aunque en la tabla resumen podemos observar los datos usando ya las medias obtenidas de la tabla anterior:

POPCOUNT	-O0	-O1	-O2
popcount1	1168344	233376,1	231002,1
popcount2	1259070,3	354347,9	283551,1
popcount3	400973,8	321509,6	356015,6
popcount4	313489,2	87210,1	88468,3
popcount5	13080,6	6947,5	7339,8
popcount5V2	3484,2	233,8	274,7
popcount6	43155,8	7814,9	7892,8
popcount7	26653,6	15221,4	15242,5

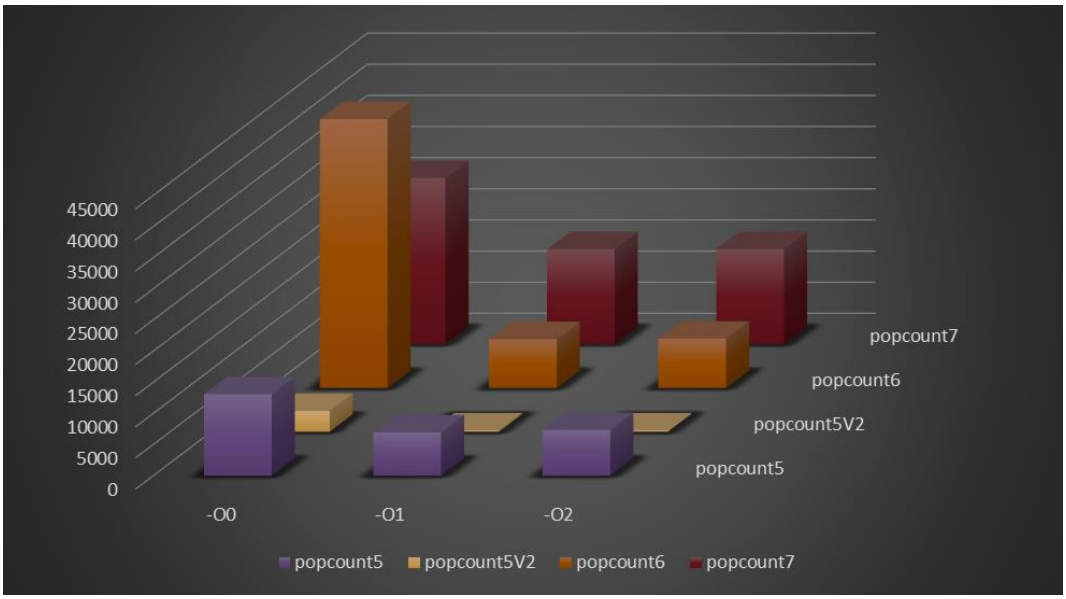
Como vemos por regla general se produce una mejora en a mayor optimización, notándose algo menos entre o1 y o2. Tenemos un claro ganador, se trata de la versión 2 de popcount 5, en la que se ahorra mucho tiempo evitando acceso a memorias innecesarias.

Nota: Esta versión se realizó en clase de prácticas el día 22 a raíz de una duda sobre los accesos a memoria de popcount5, al ver que mejoraba notablemente los tiempos obtenidos en el resto de popcounts, se ha decidido meterla en la práctica para su estudio.

A continuación podemos observar gráficamente los resultados obtenidos para poder sacar conclusiones de una manera más rápida gracias al elemento gráfico.



En general podemos ver que la O0 de casi todos los popcount va peor que las siguientes, no obstante existe un salto de calidad elevado entre las soluciones ofrecidas por los primeros 4 popcount y los 4 últimos, así que en la siguiente grafica se podrán observar de forma más detallada.



Aquí podemos ver que los accesos a memoria de popcount 5, hacían que sus resultados no fueran tan buenos como cabría esperar, al reducirlos ha mejorado notablemente sus tiempos y se posiciona como opción favorita.

Dejando esta variante a parte, la versión 5 ofrece buenos resultados, no tan alejados de la versión 6 al menos en las optimizaciones 1 y 2, la O0 sin duda es la más dispar entre todas siendo popcount6 la peor parada. No obstante la versión 7 es claramente la peor en la mayoría de casos, siendo aun así mejor que sus antecesoras.

PARITY

Después de realizar las modificaciones a los códigos y obtener las salidas al igual que en el caso anterior, se ha obtenido una tabla de datos, una media de los mismos y varias graficas que ayudan a ver mejor que variantes de parity son mejores.

Optimización -O0	0	1	2	3	4	5	6	7	8	9	10	Media
parity1 (en lenguaje C - for)	1198090	1183554	1194375	1195406	1189897	1187723	1197965	1197292	1208959	1192787	1209036	1195699,4
parity2 (en lenguaje C - while)	1269312	1275084	1260807	1268569	1263566	1261637	1276358	1271625	1281780	1279228	1274002	1271265,6
parity3 (CS:APP 3.22 - mask final)	1266135	1261853	1257833	1254209	1248426	1249904	1261320	1256375	1258574	1266730	1253141	1256836,5
parity4 (lenguaje ASM - cuerpo while)	318262	317078	316962	318389	316484	319792	319554	319814	316548	325797	316791	318720,9
parity5 (CS:APP 3.49 - 32b.16.1b)	173103	172970	173093	173573	173076	174457	174600	176589	176569	188400	172729	175605,6
parity6 (eng ASM - cuerpo for - setmp)	44397	44465	44680	44498	47120	44891	44944	44952	44860	45063	44439	44991,2

Optimización -O1	0	1	2	3	4	5	6	7	8	9	10	Media
parity1 (en lenguaje C - for)	419737	450191	435676	427219	424475	420227	423869	428497	418328	427614	419591	427568,7
parity2 (en lenguaje C - while)	351335	351528	352446	351011	351661	349108	351644	352797	354061	353599	350019	351787,4
parity3 (CS:APP 3.22 - mask final)	285775	261933	263253	265617	264586	260522	258297	274017	274737	263055	264124	265014,1
parity4 (lenguaje ASM - cuerpo while)	246636	253370	251001	250153	248894	248410	247420	247246	254418	251188	256448	250854,8
parity5 (CS:APP 3.49 - 32b.16.1b)	67383	72051	69424	69762	68951	68970	67626	67318	68255	64918	66777	68405,2
parity6 (eng ASM - cuerpo for - setmp)	13973	18781	14435	14273	13960	13758	17633	14005	13692	15194	14460	15019,1

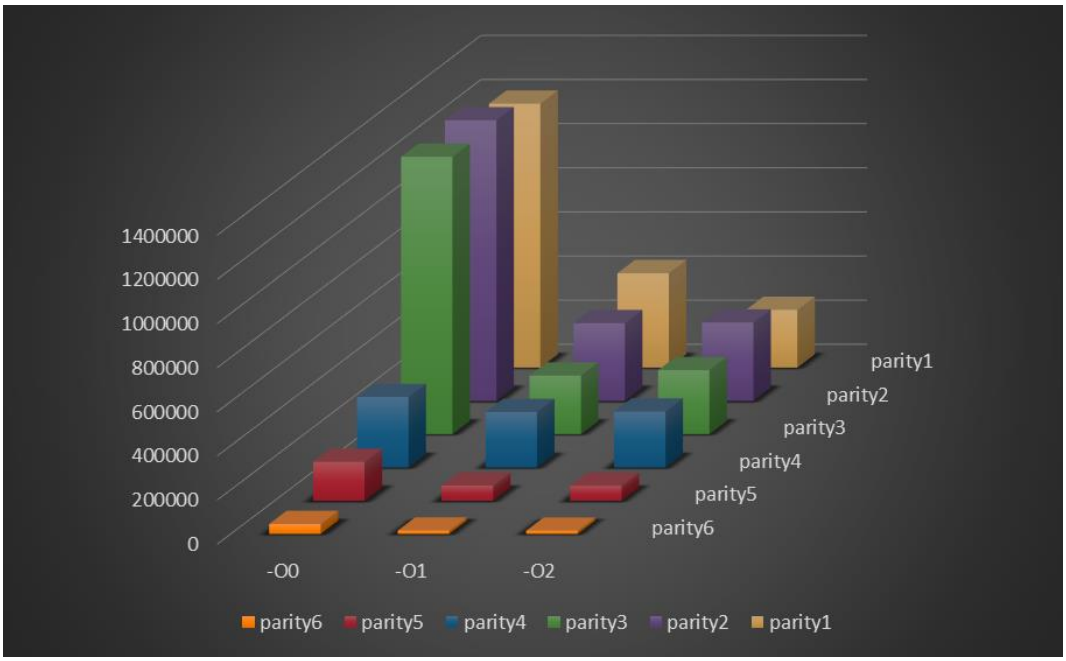
Optimización -O2	0	1	2	3	4	5	6	7	8	9	10	Media
parity1 (en lenguaje C - for)	273123	247661	263384	263482	259581	267102	257170	257933	275791	265695	255706	261350,5
parity2 (en lenguaje C - while)	368045	349041	366447	357178	350865	358094	348023	354506	364482	352570	347906	354911,2
parity3 (CS:APP 3.22 - mask final)	283815	292014	289675	291135	289721	283764	282320	288929	303337	289346	286621	289686,2
parity4 (lenguaje ASM - cuerpo while)	245227	263218	250545	263999	251638	249893	245940	246091	258883	246549	252155	252891,1
parity5 (CS:APP 3.49 - 32b.16.1b)	66669	67193	71131	66748	68183	66013	65161	66973	68145	67840	68132	67551,9
parity6 (eng ASM - cuerpo for - setmp)	14156	14130	22874	14134	13701	13346	13341	14179	13450	13911	14077	14714,3

A continuación la tabla resumen de las medias:

PARITY	-O0	-O1	-O2
parity1	1195699	427569	261351
parity2	1271266	351787	354911
parity3	1256837	265014	289686
parity4	318721	250855	252891
parity5	175606	68405,2	67551,9
parity6	44991,2	15019,1	14714,3

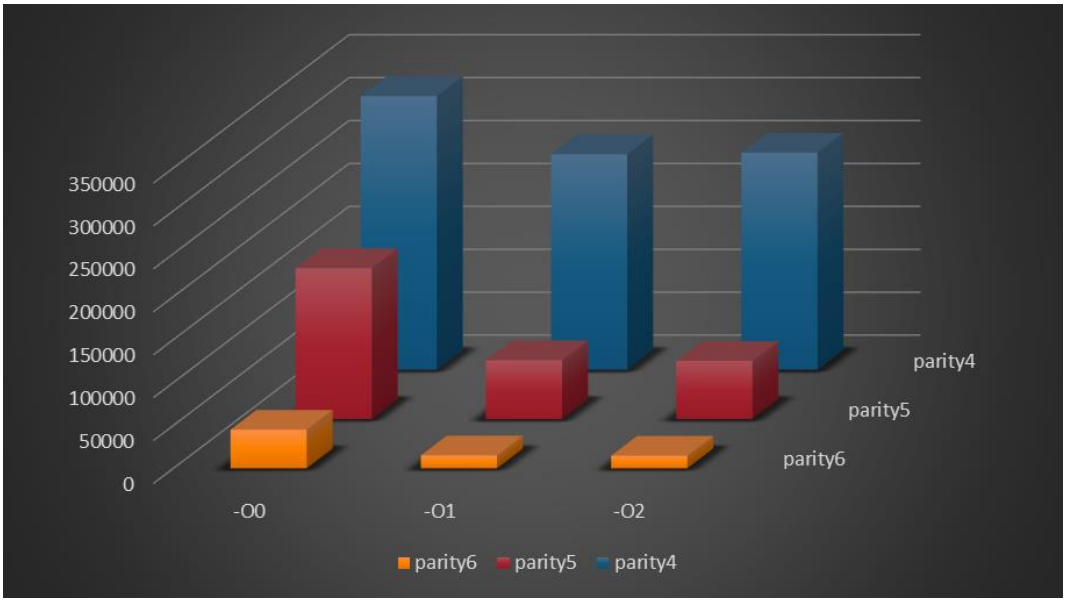
En este caso la amalgama de números es más complicada de ver que en el caso de popcount, aunque ocurre algo parecido, de la versión 1 a la 4 los datos son algo peores que en las siguientes, no obstante esto tiene un pequeño matiz que se explicará a continuación ya que se puede observar de mejor forma gracias a las gráficas.

En primer lugar tenemos la tabla de comparativa de todas las versiones:



Aquí podemos observar que parity6 es claramente mejor en todos los modos de optimización, no obstante de la 1 a la 4, las versiones optimizadas de O2 son bastante similares, las otras optimizaciones tienen alguna diferencia más pero son también bastante parecidas, obviando la 4º, que desde O0 produce unos resultados bastante superiores a las versiones anteriores.

A continuación una versión ampliada de las versiones 4, 5 y 6:



Poco que añadir, parity6 arrasa en rendimiento con sus hermanas próximas, pero las mejoras entre O1 y O2 son casi inapreciables por lo que no parecen suponer una mejora real en los tiempos de ejecución.

Cuestionarios

Cuestiones sobre Popcount

1.- Para entero 2^{26} -1, para unsigned es 2^{27}-1

2.- Utilizando el bit 0 y usándolo como bit a comparar, podemos ver si es un 1 y utilizarlo para contar, posteriormente se pueden desplazar todos los números hasta el último de ellos.

El resto de bits, no se podrían utilizar, salvo el 31 si utilizamos un desplazamiento hacia la izquierda.

3.- Se utiliza para que los desplazamientos no afecten al signo. Si se declara como int, el primer signo es el que indica el signo, así que al desplazarlo cambiaria.

En principio no supone diferencia para nuestra práctica, pero en caso de necesitar números negativos o el máximo valor alcanzable por un entero (sin signo) no podríamos realizar la cuenta bien.

4.- Los tiempos usando m y +m son más grandes, esto se debe a que los valores se pasan a través de la pila y esto supone más accesos que pasarlos por registro.

5.- Se trata de la necesidad de almacenar los vectores de forma entera, para 1 no habría problema pero al ser 2 se necesitarían más registros de los que se disponen, por esto se hace uso de la memoria en su lugar, ya que tiene mas "capacidad".

8.- Esta pregunta se ha respondido con los gráficos de arriba.

Cuestiones sobre Parity

2.- Se tendría en cuenta el bit de signo para la paridad en caso de no usarte el unsigned, por lo que no sería acertado.

3.- Esta comprobación se ha realizado con los gráficos anteriores.

4.- Al igual que la pregunta anterior se ha detallado en los gráficos anteriores.

5.- Los tiempos usando m y +m son más grandes, esto se debe a que los valores se pasan a través de la pila y esto supone más accesos que pasarlos por registro.

9.- Realizado mas arriba.